



Ektron® eWebEditPro+XML Developer's Reference Guide

Release 5.1, Revision 1

Ektron® Inc.
5 Northern Blvd., Suite 6
Amherst, NH 03031
Tel: +1 603-594-0249
Fax: +1 603-594-0258
Email: sales@ektron.com
<http://www.ektron.com>

Copyright June 2007 Ektron®, Inc. All rights reserved.

EKTRON is a registered trademark of Ektron, Inc.

Release 5.1, Revision 1, June 2007

EKTRON, INC. SOFTWARE LICENSE AGREEMENT

** READ THIS BEFORE LOADING SOFTWARE**

YOUR RIGHT TO USE THE PRODUCT ENCLOSED IN THIS ENVELOPE OR OTHERWISE DELIVERED IS SUBJECT TO THE TERMS AND CONDITIONS SET OUT IN THIS LICENSE AGREEMENT. OPENING THIS ENVELOPE OR USING THIS PRODUCT SIGNIFIES YOUR AGREEMENT TO THESE TERMS. IF YOU DO NOT AGREE TO THIS SOFTWARE LICENSE AGREEMENT, YOU MAY RETURN THE PACKAGE WITH THE UNOPENED ENVELOPE OR AS IT WAS DELIVERED AND THE UNDAMAGED SOFTWARE ENCLOSED, ALONG WITH THE RECEIPT, TO YOUR SUPPLIER OR TO EKTRON, INC. WITHIN THIRTY DAYS FROM THE DATE OF PURCHASE FOR A FULL REFUND.

CUSTOMER should carefully read the following terms and conditions before using the software program(s) contained herein (the Software). Opening this sealed envelope, and/or using the Software or copying the Software onto CUSTOMER'S computer hard drive indicates CUSTOMER'S acceptance of these terms and conditions. If CUSTOMER does not agree with the terms of this agreement, CUSTOMER should promptly return the unused and unopened Software for a full refund.

Ektron, Inc. (Ektron) grants, and the CUSTOMER accepts, a nontransferable and nonexclusive License to use the Software on the following terms and conditions:

1. Right to use: The Software is licensed for use only in delivered code form. Each copy of the Software is licensed for use only on a single URL. Each license is valid for the number of seats listed below (the Basic Package). Any use of the Software beyond the number of authorized seats contained in the Basic Package without paying additional license fees as provided herein shall cause this license to terminate. This is not a concurrent use license. Should CUSTOMER wish to add seats beyond the seats licensed in the Basic Package, the CUSTOMER may add seats on a block basis at the then current price for additional seats (see product pages for current price). The Basic Packages are as follows:

eWebEditPro - Licensed for ten (10) seats per URL.

For purposes of this section, the term seat shall mean an individual user provided access to the capabilities of the Software.

2. Duration: This License shall continue so long as CUSTOMER uses the Software in compliance with this License. Should CUSTOMER breach any of its obligations hereunder, CUSTOMER agrees to return all copies of the Software and this License upon notification and demand by Ektron.

3. Copyright: The Software (including any images, applets, photographs, animations, video, audio, music and text incorporated into the Software) as well as any accompanying written materials (the Documentation) is owned by Ektron or its suppliers, is protected by United States copyright laws and international treaties, and contains confidential information and trade secrets. CUSTOMER agrees to protect the confidentiality of the Software and Documentation. CUSTOMER agrees that it will not provide a copy of this Software or Documentation nor divulge any proprietary information of Ektron to any person, other than its employees, without the prior consent of Ektron; CUSTOMER shall use its best efforts to see that any user of the Software licensed hereunder complies with this license.

4. Limited Warranty: Ektron warrants solely that the medium upon which the Software is delivered will be free from defects in material and workmanship under normal, proper and intended usage for a period of three (3) months from the date of receipt. Ektron does not warrant the use of the Software will be uninterrupted or error free, nor that program errors will be corrected. This limited warranty shall not apply to any error or failure resulting from (i) machine error, (ii) Customer's failure to follow operating instructions, (iii) negligence or accident, or (iv) modifications to the Software by any person or entity other than Company. In the event of a breach of warranty, Customer's sole and exclusive remedy, is repair of all or any portion of the Software. If such remedy fails of its essential purpose, Customer's sole remedy and Ektron's maximum liability shall be or refund of the paid purchase price for the defective Products only. This limited warranty is only valid if Ektron receives written notice of breach of warranty within thirty days after the warranty period expires. In the event of a breach of warranty, Ektron's sole responsibility, and CUSTOMER'S sole and exclusive remedy, is correction of any defect or bug causing the breach of warrant (either by repair or replacement of the Software). In the event this remedy fails of its essential purpose, CUSTOMER's sole and exclusive remedy shall be refund of the Purchase Price of the defective Software only. This limited warranty is only valid if Ektron receives written notice of breach of warranty within thirty days following the warranty period.

5. Limitation of Warranties and Liability: THE SOFTWARE AND DOCUMENTATION ARE SOLD AS IS AND WITHOUT ANY WARRANTIES AS TO THE PERFORMANCE, MERCHANTABILITY, DESIGN, OR OPERATION OF THE SOFTWARE. BECAUSE OF THE DIVERSITY OF CONDITIONS UNDER WHICH THIS PRODUCT MAY BE USED, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. EXCEPT AS DESCRIBED IN SECTION 4, ALL WARRANTIES EXPRESS AND IMPLIED ARE HEREBY DISCLAIMED.

THE REMEDY DESCRIBED IN SECTION 12 SHALL BE CUSTOMER'S SOLE REMEDY FOR ANY PERFORMANCE FAILURE OF THE PRODUCTS. NEITHER COMPANY NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS OR GOODWILL, LOSS OF DATA OR USE OF DATA, INTERRUPTION OF BUSINESS NOR FOR ANY OTHER INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND UNDER OR ARISING OUT OF, OR IN ANY RELATED TO THIS AGREEMENT, HOWEVER, CAUSED, WHETHER FOR BREACH OF WARRANTY, BREACH OR REPUDIATION OF CONTRACT, TORT, NEGLIGENCE, OR OTHERWISE, EVEN IF COMPANY OR ITS REPRESENTATIVES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS.

6. Miscellaneous: This License Agreement, the License granted hereunder, and the Software may not be assigned or in any way transferred without the prior written consent of Ektron. This Agreement and its performance and all claims arising from the relationship between the parties contemplated herein shall be governed by, construed and enforced in accordance with the laws of the State of New Hampshire without regard to conflict of laws principles thereof. The parties agree that any action brought in connection with this Agreement shall be maintained only in a court of competent subject matter jurisdiction located in the State of New Hampshire or in any court to which appeal therefrom may be taken. The parties hereby consent to the exclusive personal jurisdiction of such courts in the State of New Hampshire for all such purposes. The United Nations Convention on Contracts for the International Sale of Goods is specifically excluded from governing this License. If any provision of this License is to be held unenforceable, such holding will not affect the validity of the other provisions hereof. Failure of a party to enforce any provision of this Agreement shall not constitute or be construed as a waiver of such provision or of the right to enforce such provision. If you fail to comply with any term of this License, YOUR LICENSE IS AUTOMATICALLY TERMINATED. This License represents the entire understanding between the parties with respect to its subject matter.

Esker Active X Plug-in, Version 4.4

Active X controls under Netscape

Use License

=====

IMPORTANT: READ CAREFULLY -

Use of the Esker Active X Plug-in, Version 4.4, is subject to the terms and conditions below. BY INSTALLING, COPYING OR OTHERWISE USING THE PLUG-IN, YOU AGREE TO BE BOUND BY THE TERMS AND CONDITIONS BELOW. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, DO NOT INSTALL, COPY OR USE THE PLUG-IN.

The Plug-in is provided to you as an end-user "as is" without technical support. No rights are granted to you in this license for commercial use or redistribution of any kind. Should you desire to redistribute the Plug-in or include it with other software packages please e-mail Esker at axplug-in@esker.com to find out how you may do so.

DISCLAIMER OF WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ESKER AND ITS SUPPLIERS PROVIDE TO YOU THE PLUG-IN AS IS AND WITH ALL FAULTS; AND ESKER AND ITS SUPPLIERS HEREBY DISCLAIM WITH RESPECT TO THE PLUG-IN ALL WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) WARRANTIES OR CONDITIONS OF OR RELATED TO: TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR COMPLETENESS OF RESPONSES, RESULTS, LACK OF NEGLIGENCE OR LACK OF WORKMANLIKE EFFORT, QUIET ENJOYMENT, QUIET POSSESSION, AND CORRESPONDENCE TO DESCRIPTION. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE PLUG-IN REMAINS WITH YOU.

EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ESKER OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER INCLUDING, BUT NOT LIMITED TO,

DAMAGES FOR: LOSS OF PROFITS, LOSS OF CONFIDENTIAL OR OTHER INFORMATION, BUSINESS INTERRUPTION, PERSONAL INJURY, LOSS OF PRIVACY, FAILURE TO MEET ANY DUTY (INCLUDING OF GOOD FAITH OR OF REASONABLE CARE), NEGLIGENCE, AND ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE PLUG-IN, OR FAILURE TO PROVIDE TECHNICAL SUPPORT, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS LICENSE, EVEN IF ESKER OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Due to the complex nature of computer software Esker does not warrant that the Plug-in is completely error-free, will operate without interruption or is compatible with all equipment and software configurations. You are advised to check all work performed with the Plug-in. Do not use the Plug-in in any case where significant damage or injury to persons, property or business may happen if an error occurs. You expressly assume all risks for such use.

© 2001 Esker, Inc. All rights reserved. Copyright to and in the Plug-in remains the property of Esker, Inc. and as such, any copyright notices in the Plug-in are not to be removed.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, THAT YOU UNDERSTAND THIS AGREEMENT, AND UNDERSTAND THAT BY CONTINUING THE INSTALLATION OF THE SOFTWARE, BY LOADING OR RUNNING THE SOFTWARE, OR BY PLACING OR COPYING THE SOFTWARE ONTO YOUR COMPUTER HARD DRIVE, YOU AGREE TO BE BOUND BY THIS AGREEMENT'S TERMS AND CONDITIONS. YOU FURTHER AGREE THAT, EXCEPT FOR WRITTEN SEPARATE AGREEMENTS BETWEEN EKTRON AND YOU, THIS AGREEMENT IS A COMPLETE AND EXCLUSIVE STATEMENT OF THE RIGHTS AND LIABILITIES OF THE PARTIES.

(c) 1999-2003 Ektron, Inc. All rights reserved. LA10031, Revision 1.5b

Summary Table of Contents

Introduction	1
eWebEditPro+XML Object Model.....	2
eWebEditPro+XML API Cheat Sheet.....	29
Commands.....	195
Using JavaScript to Send Commands	197
Standard Commands	199
Custom Commands	215
Using eWebEditPro+XML	224
eWebEditPro+XML Dataflow	226
Defining the Toolbar	230
Dynamically Changing the Editor.....	250
Customizing the Popup Button	253
Customizing Context Menus	256
Modifying the Language of eWebEditPro+XML.....	265
Customizable JavaScript Files.....	291
Client Installation Pages	297
JavaScript Objects	300
ActiveX Control	309
The Configuration Data.....	312
Encoding Special Characters.....	417
Style Sheets.....	430
Managing Hyperlink Dialogs	445
Managing Images	455

Content Upload	563
WebImageFX	577
eWebEditPro+XML's XML Features	600
Supporting the Data Designer.....	601
Original XML Functionality	644
Validating XHTML Content.....	727
Integrating eWebEditPro+XML.....	734
Appendices	778
Appendix A: Naming the eWebEditPro+XML Editor	778
Appendix B: Error Messages	779
Appendix C: eWebEditPro+XML Architecture	785
Appendix D: Automatic Upload File Types.....	787

Detailed Table of Contents

Introduction	1
eWebEditPro+XML Object Model.....	2
ewebeditproevents Object	3
eWebEditProUtil Object	4
eWebEditPro Object	4
Event Object	6
Parameters Object	7
Popups Object	8
Instances Object	9
InstallPopup Object.....	10
Popup Object	11
Button Tag Object	12
Image Tag Object.....	12
eWebEditPro ActiveX Control Object.....	13
Image Editor Object	16
XML Object	18
Toolbars Object	19
Media File Object.....	20
XML Data Object.....	23
Command Item Object	25
Automatic Upload Object	27
eWebEditPro+XML API Cheat Sheet.....	29
Alphabetical List of Methods, Properties and Events.....	29
Master List of Methods.....	50
Master List of Properties	134
Master List of Events	182
Commands.....	195
How Commands are Processed	195
Sources of Commands	196
Using JavaScript to Send Commands	197
Sending Commands Programmatically.....	197
Standard Commands	199
Standard Command Guidelines	199

List of Standard Commands.....	199
Table Commands.....	207
The cmdfontcolorvalue Command.....	207
Find and Replace and Find Next Commands.....	209
Special Character Commands	212
Custom Commands	215
Custom Command Guidelines	215
Creating a Custom Command	215
Defining a Custom Function for All Occurrences of the Editor	216
Writing a Custom JavaScript Event Function for One or More Occurrences of the Editor.....	220
Detecting When a Standard Command is Executed	222
Defining a Custom JavaScript Function	222
Using eWebEditPro+XML	224
Design and Implementation Guidelines	224
System Requirements	224
Maximum Size of Content	225
Placing More Than One Editor on a Page.....	225
Samples.....	225
Memory Considerations.....	225
Recommendations.....	225
eWebEditPro+XML Dataflow	226
Integrating eWebEditPro+XML into a Web Page.....	226
Content Flow Diagram	226
1. The Edit Page: Read Content.....	227
2. The Hidden Field	228
3. The onload Event	228
4. The onsubmit Event.....	228
5. The Action Page: Write Content	229
Defining the Toolbar.....	230
Modifying Configuration Data.....	230
Toolbar Menus	230
Defining the eWebEditPro+XML Toolbar	231
Determining Which Menus Appear on the Toolbar	231
Finding a Toolbar Menu's Internal Name	232
Creating a Custom Toolbar Menu.....	233
Removing a Toolbar Menu.....	234
Removing All Toolbars.....	234
Placing a Toolbar Menu on a Row with Another Menu.....	235
Determining if a Toolbar Menu Should Wrap to the Next Row	235
Creating or Editing the Toolbar Menu Caption.....	236

Determining Which Buttons and Dropdown Lists Appear on a Menu ..	237
Adding a Toolbar Button	237
Adding a Dropdown List	238
Removing a Toolbar Button or Dropdown List	240
Rearranging Buttons/Dropdown Lists on a Toolbar Menu	240
Adding a Space Between Two Toolbar Menu Items	241
Adding a Separator Bar Between Two Toolbar Menu Items	241
Changing the Image that Appears on a Toolbar Button.....	242
Displaying Button Caption Text.....	243
Defining the Alignment of Caption Text	243
Translating Button Captions and Tool Tips	244
Creating a Popup Menu	245
Determining which Fonts, Font Sizes, and Headings are Available.....	246
Changing Available Fonts.....	247
Changing Available Font Sizes.....	248
Changing Available Headings	248
Creating a List Item that Generates No Command	249
Dynamically Changing the Editor	250
Dynamically Creating Configuration Data on the Server Side	250
Avoiding Problems When Dynamically Changing the Toolbar on the Server	251
Dynamically Changing the Editor on the Client Using JavaScript.....	251
Disabling and Enabling Menu Items within Scripting	251
Accessing Menus and Commands	251
Enabling and Disabling a Command	252
Customizing the Popup Button.....	253
Customizing the createButton Command	254
Customizing Context Menus	256
Removing Commands from a Context Menu.....	256
Context Menu Commands and their Internal Names	257
Suppressing the Context Menu.....	257
The Toolbar Object Interface	258
Defining Menus and Commands.....	258
Toolbar Object Quick Reference	258
Command Object Quick Reference	258
Script Example	259
Command Values.....	259
etbToolbarOptions	259
etbToolbarStyles	260
etbCaptionAlignment	260
etbToolbarLocation	261
etbToolbarModifications.....	261
etbCommandOptions.....	262

etbCommandStyles	262
etbCommandModifications	263
etbErrorValues	263
Modifying the Language of eWebEditPro+XML.....	265
How eWebEditPro+XML Determines the User Interface Language ..	265
Locale Files.....	266
Standard Locale Files	266
Translating eWebEditPro+XML's User Interface.....	267
Displaying the User Interface in a Translated Language.....	268
Translating the User Interface to a Windows-Supported Language	269
Languages Supported by Windows	279
Terms on the Supported Languages Table.....	279
Working with non-English Content.....	284
Accented Characters	285
Using the Languages Sample	285
Displaying Menus and Dialogs in a non-European Language	285
Setting the Language of Spell Checking	287
Modifying Standard Text (including English)	287
Location of Translated Strings	288
Modifying American English Text.....	289
Modifying the Standard Text of a Translated Language	289
Modifying the Standard Text of a Windows-Supported Language .	290
Customizable JavaScript Files.....	291
The ewebeditpro.js File	291
The ewebeditprodefaults File	291
The ewebeditpromessages File	292
Disabling the "Click OK to Preserve Changes" Message	294
The ewebeditproevents File	295
The ewebeditpromedia File.....	296
Client Installation Pages.....	297
Customizing the Client Installation Pages	298
Disabling the Installation Pages	299
What Happens When Auto Install Fails or is Cancelled.....	299
JavaScript Objects	300
The JavaScript Object Model	300
JavaScript Object Properties, Methods and Events	300
Event Handler Functions.....	300
Double-Click Element Handlers	301
The eWebEditProExecCommandHandlers Array	301
ExecCommandHandlersArray Parameters	302

Parameter Requirements for Commands	302
The Toolbar Reset Command	303
Reacting to the Initialization of a Toolbar	303
When the Event is Sent to the Script.....	303
Using ToolbarReset to Reset Customization	304
The Redisplay Toolbars Command.....	304
The Instance Object.....	304
The onerror Event.....	305
The instanceTypes Array	305
The Parameters Object.....	306
Parameters Object Properties	306
Installation Popup Window Defaults	307
Popup Window Defaults	307
eWebEditProUtil JavaScript Object.....	307
ActiveX Control	309
Accessing the ActiveX Control Using JavaScript.....	309
eWebEditPro+XML JavaScript object	309
eWebEditPro+XML ActiveX control.....	309
Instance JavaScript object.....	310
ActiveX Properties, Methods and Events.....	310
The Configuration Data.....	312
Managing the Configuration Data	312
Editing the Configuration Data.....	312
Providing Configuration Files for User Groups	313
Changing the Configuration Data's Location	314
Troubleshooting Problems with the Configuration Data	314
Organization of Configuration Documentation	314
Letting Users Customize the Toolbar	315
Allowing User Customization	316
Preventing Customization by Users.....	316
Overriding User Customization	317
Determining Which Configuration Data to Use	317
Changes to config.xml Have No Effect	317
Overview of Configuration Data	319
Configuration Data: Functional View	320
Configuration Data: Functional View Topic List	321
Configuration Data: Hierarchical View	322
Configuration Elements in Alphabetical Order	322
The Config Element	326
The Interface Element.....	326
Buttons not Assigned to Menus.....	327
The Features Element	327
Attribute Types	327
Boolean	327

Integer	328
String	328
User Interface Elements: Standard, Menu, and Popup	329
User Interface Element Hierarchy	330
User Interface Elements in Alphabetical Order	331
User Interface Element Definitions	332
bar	332
button.....	333
Caption	335
command.....	336
cmd.....	339
config.....	340
features.....	341
image.....	342
interface.....	343
listchoice.....	345
menu.....	349
popup.....	351
selections.....	352
space	353
standard.....	354
style	357
toolTipText	358
Button Images.....	360
Formats Supported	360
Sources of Images	360
Images Supplied by eWebEditPro+XML.....	360
Creating Your Own Images	369
Image File Extensions	369
Size of Button Images	369
Background Color of Button Images.....	370
Button Image Specification Summary.....	370
Managing Tables	371
The Table Element of the Configuration Data	371
Element Hierarchy	371
Child Elements	371
Attributes	371
Allowing Users to Create Tables	372
Customizing the Table Dialogs.....	372
Restricting Table Options	374
Customizing the Tables Menu	375
Customizing the Tables Toolbar Menu	376
Setting Default Values for the Insert Table Dialog.....	377
Controlling Alignment Field Responses	378
Controlling Responses for the Horizontal Alignment Field	378
Controlling Responses for the Vertical Alignment Field.....	379
Fonts and Headers	380

fonts	380
Element Hierarchy	380
Attributes	380
fontname	380
Remarks	381
Element Hierarchy	381
Attributes	381
fontsize.....	381
Remarks	381
Element Hierarchy	382
Attributes	382
headings	382
Element Hierarchy	383
Attributes	383
heading[x]	383
Remarks	383
Element Hierarchy	384
Attributes	385
External Features.....	386
Description	386
Element Hierarchy	386
Attributes	386
Adding External Features	386
Examples.....	386
Form Elements.....	388
Description.....	388
Element Hierarchy	388
Attributes	388
Viewing and Editing HTML Content	390
The ViewAs Feature	390
Disabling Custom Toolbar Buttons View as HTML Mode	390
The EditHTML Feature	392
Cleaning HTML.....	393
Clean Element Hierarchy	393
Providing User Access to the Clean Feature	393
Clean Element	394
Element Hierarchy	394
Child Elements	394
Attributes	395
Remove Element	399
Element Hierarchy	399
Child Elements	399
Attributes	400
Endtag Element	400
Element Hierarchy	400
Attribute	400
Example.....	400

Attribute Element	401
Element Hierarchy	401
Attribute	401
Example.....	401
Tagonly and Tagelement Elements	401
Element Hierarchy	401
Attribute	402
Example.....	402
TagWoAttr Element	402
Element Hierarchy	402
Attribute	402
Example.....	402
xsltFilter Element	403
The Spellcheck Feature.....	404
Spellcheck.....	405
Element Hierarchy	405
Child Elements	405
Attributes	405
Spellayt	406
Element Hierarchy	407
Attributes	407
Spellingsuggestion	407
Element Hierarchy	408
Attributes	408
Example of Spell Check Features.....	408
The Math Expression Editor Feature	409
Math	409
Element Hierarchy	409
Child Elements	409
Attributes	409
Toolbar	410
Element Hierarchy	410
Attributes	410
Example of Math Features.....	410
Editing in Microsoft Word	411
Element Hierarchy	411
Child Elements	411
Attributes	412
Using the Long Parameter with cmdmsword.....	412
How Microsoft Word Content is Processed.....	412
Conserve Word Formatting	413
Convert Styles	414
Conform by Discarding	415
Options	415
Using Word to Edit XML Documents.....	415
Encoding Special Characters.....	417

Factors that Affect the Display of Special Characters	417
Viewing and Saving Unicode Characters	418
Displaying Asian Languages	419
Unicode Characters.....	419
Configuring for Extended and Special Characters	419
charencode Attribute	419
Choosing a charencode Value.....	422
Character Encoding Checklist.....	424
UTF-8	425
How to Store Unicode Characters So They Are Searchable	425
References	426
Implementing a Web Site that Uses UTF-8 Encoding..	427
Implementing UTF-8	427
Tips	428
Setting the charset Parameter.....	428
Browser Support for UTF-8	428
For More Information about UTF-8.....	428
Style Sheets	430
Using Style Sheets to Standardize Formatting.....	431
The Default Style Sheet	431
Changing the Default Style Sheet.....	431
Applying Style Sheets	431
Specifying a Style Sheet in the Configuration Data	432
Adding a Style Sheet to a Single Page.....	432
Dynamically Changing a Style Sheet for a Single Instance of the Editor	433
The BodyStyle Parameter	433
Preserving Tags When Office Content is Pasted	433
Saving Style Sheet Tags When Content is Saved	434
Setting Publishstyles to True	434
Setting Publishstyles to False.....	434
Inserting span or div Tags	434
Applying Two Style Classes to the Same Content	436
Location of equivClass Attribute	436
How the Editor Determines if Two Classes Are Equivalent	436
New Class is Equivalent to Original Class.....	437
New Class is not Equivalent to Original Class.....	437
Forcing Two Classes to be Equivalent	438
Tips for Using this Feature.....	438
Implementing Style Class Selectors.....	439
Example of Using Style Class Selectors.....	439
Types of Style Classes	440
Determining Which Style Classes Appear in the Dropdown List ...	440
Determining the Names in the Dropdown List	441
Suppressing Styles from the Dropdown List.....	442
Style Classes and Matching Attributes	442

Managing Hyperlink Dialogs	445
Customizing Dropdown Lists in the Hyperlink Dialog Box	445
Customizing the Lists of the Hyperlink Dialog Box	446
Quick Link List	446
Type List	448
Target Frame List.....	450
Specifying Default Values for the Insert Hyperlink Dialog.....	452
Editing the New HyperLink Dialog Box	453
Editing Quick Links	454
Dynamically Creating the Quick Links File	454
Managing Images	455
How Image Selection Works	455
Organization of the Image Selection Documentation.....	456
Customizing the Alignment Field of the Picture Properties Dialog.....	457
Modifying Alignment Field Responses	457
Setting a Default Response for the Alignment Field.....	458
Removing the Alignment Field from the Picture Properties Dialog	458
Examples of Implementing Image Selection	459
Example 1: No Restrictions, No Saving to a Database	459
Example 2: File Size Restriction, No Saving to Database	462
Example 3: FTP	466
Example 4: Database Samples	470
Implementing Image Upload	472
FTP File Upload.....	472
HTTP File Upload	473
Using EktronFileIO for Your Own Image Uploads	478
Step 1: Create a Selection Web Page	478
Step 2: Create a Form with a File Selection Field Item	479
Step 3: Creating an ASP Page to Activate the Posted Upload	480
Step 4: Providing Upload Feedback	481
ColdFusion	484
Manipulating Media File Methods and Properties	486
Using Local or Given Image Path Resolutions	486
Base URL	487
Given Resolution Type	487
Programmatically Accessing Media File Properties	488
Accessing the Media File Object	488
Using Netscape to Access Image Properties	488
Entry Point for Using External Scripts	489
Setting External Page Parameters	490
Changing the Transfer Method on the Fly	490
Programmatically Changing from the Default of FTP to the ASP Library.....	491
Specifying an Image to Insert	491
Modifying the Upload Directory	492

The Mediafiles Feature	493
Mediafiles Element Hierarchy	493
User Interface Elements in Alphabetical Order	494
Mediafiles Element.....	495
Description.....	495
Element Hierarchy	495
Child Elements	495
Attributes	495
Validext Element	495
Description.....	495
Element Hierarchy	495
Attributes	496
Example.....	496
Maxsizek Element.....	496
Description.....	496
Element Hierarchy	496
Attributes	496
Mediaconfig Element	496
Description.....	496
Element Hierarchy	496
Attributes	497
Example.....	497
Transport Element.....	497
Description.....	497
Element Hierarchy	497
Child Elements	497
Attributes	498
Autoupload Element.....	498
Description.....	498
Element Hierarchy	498
Attributes	499
Username Element	502
Description.....	502
Element Hierarchy	502
Attributes	502
Password Element	502
Description.....	502
Element Hierarchy	502
Attributes	503
Proxyserver Element.....	503
Description.....	503
Element Hierarchy	503
Attributes	503
Domain Element	503
Description.....	503
Element Hierarchy	503
Attributes	504
Xferdir Element	504

Description.....	504
Element Hierarchy	504
Attributes	505
Webroot Element	505
Description.....	505
Element Hierarchy	505
Attributes	506
Defsource Element	506
Description.....	506
Element Hierarchy	506
Attributes	506
Port Element	506
Description.....	506
Element Hierarchy	506
Attributes	507
Resolvemethod Element.....	507
Description.....	507
Element Hierarchy	507
Attributes	508
Imageedit element	508
Description.....	508
Element Hierarchy	508
Child Elements	508
Control Element	508
Description.....	508
Element Hierarchy	508
Attributes	509
Setting up an Image Repository.....	510
The Image Repository Folder	510
Inserting an Image into a Web Page.....	511
Example.....	512
Dynamically Selecting Upload Destinations	513
Setting Up Image Upload.....	513
Media File Object.....	514
Modifying the Upload Location	514
Sample HTML Page	515
User Selection – Changing the Upload Location.....	516
Full Example.....	517
Automatic Upload	520
Automatic Upload of Files and Images from an External Application ..	520
Installing the Automatic Upload Feature	522
Modules that Enable Automatic Upload	522
An Example of Customizing Automatic Upload	522
cmdmfuploadall Command	523
Overview of the Automatic Upload Process.....	523
The Upload Process	523
Information Components.....	525

Concepts	525
eWebEditPro+XML Fields Sent with Post.....	526
Image Upload Fields.....	526
Custom Field Set.....	529
Example HTML Form	529
Creating an Automatic File Receive Script	529
What This Section Covers	530
What This Section Does Not Cover	530
The Automatic Upload Server-Side Receiving Module	530
Steps to Receiving a File	530
Step 1 – Act on the Command.....	531
Step 2 – Extract the File Information	531
Step 3 – Determine the File Destination	531
Step 4 – Extract the File Binary and Save	532
Step 5 – Build the Return XML Data.....	532
Step 6 – Respond Back to the Client.....	534
Creating the Script.....	534
Data Island	534
Steps to Receiving Content.....	539
Step 1 - Act on the Command	539
Step 2 - Extract the Content	539
Step 3 - Save the Content	539
Step 4 - Return a Response	539
EWepAutoSvr Object API	540
ClientMajorRev	540
ClientMinorRev	540
EkFileSave	540
EkFileSave2	541
EkFormFieldValue	543
EkFileSize.....	543
FileObject	544
FileCount	544
ResponseData	544
EkFileObject API.....	544
Description.....	545
FileDimensions	545
FileError.....	546
FileID	546
FileName	547
FileSize.....	547
FileType	547
FileUrl	548
Fragment	549
Thumbnail.....	549
ThumbReference	550
XML Element Descriptions.....	551
DBORDER.....	551
DESC.....	551

DHEIGHT	552
DWIDTH	552
FERROR	552
FID.....	552
FILEINFO	553
FRAGMENT	553
FSIZE	554
FSRC.....	554
FTYPE	554
FURL	555
THUMBNAIL.....	555
THUMBREF	555
UPLOAD.....	556
Image Upload Response Example with Proprietary Information.....	556
ColdFusion Example.....	557
ASP Example.....	559
Automatic Upload Object.....	562
Media File Object Properties.....	562
Automatic Upload Object Properties as a Subset of the Media Object Settings.....	562
Content Upload	563
Retrieving Content from eWebEditPro+XML	563
The Content Upload Command	563
Content Setting API	564
Automatic Upload Object Interface Properties	564
JavaScript Example	565
Fields in the Posted Form	565
Steps to Receiving Content.....	567
Step 1 - Act on the Command	567
Step 2 - Extract the Content	568
Step 3 - Save the Content	568
Step 4 - Return a Response	568
The Receiving Page.....	568
Creating a Receiving Page.....	569
Content Types.....	570
What Happens if a Content Type is Not Supported.....	570
Content Type Categories.....	570
How Content Type is Determined.....	576
WebImageFX.....	577
Using the WebImageFX Object.....	578
Assigning Configuration.....	578
Retrieving the Object	578
Checking Availability.....	578
Displaying WebImageFX	579

Controlling WebImageFX	579
Full Example.....	579
Adding a Toolbar Button to Launch WebImageFX.....	580
New Configuration Variable	580
WebImageFX's Configuration Data.....	581
fmtchange.....	582
imgcreate.....	583
imgedit.....	583
imgfmt.....	584
namechange.....	584
operations.....	585
valformats	586
valoutformats	587
Image Names	588
Specifying Image Format.....	590
Specifying Color Depth	590
Methods to Manipulate WebImageFX.....	590
Events to Manipulate WebImageFX.....	592
Commands Unique to WebImageFX	593
The IData Parameter	595
Client Script Interface for Automatic File Upload.....	596
Initializing the Automatic Upload.....	596
Interface Retrieval.....	596
Properties.....	596
AllowUpload.....	596
WebRoot.....	596
ValidExtensions	597
TransferRoot.....	597
Port	597
LoginRequired	597
LoginName	597
Password.....	597
TransferMethod	597
ServerName	597
Methods	597
GetFileDescription(FileName)	597
SetFileDescription(FileName, Description).....	597
ReadResponseHeader().....	598
AddNamedData(FileName, DataName, DataValue).....	598
ReadNamedData(FileName, DataName)	598
RemoveNamedData(FileName, DataName)	598
GetFileStatus(FileName)	598
SetFileStatus(FileName, Status).....	598
ReadUploadResponse()	598
UploadConfirmMsg(Message, Title)	598
SetFieldValue(DataName, DataValue)	598
GetFieldValue(DataName)	598
RemoveFieldValue(DataName)	598

AddFileForUpload(LocalFileName, Description)	598
ListFilesWithStatus(Status, Delim).....	598
RemoveFileForUpload(LocalFileName)	599
Property Setting Methods	599
eWebEditPro+XML's XML Features	600
Supporting the Data Designer.....	601
Changes to eWebEditPro+XML's Configuration	601
How These Configuration Files Differ from config.xml.....	602
<Viewas> Attribute: View	602
Data Designer Commands.....	603
The <cmddsgcalc> Command.....	605
Controlling the Presentation XSLT	605
The Validate Commands	606
Retrieving and Loading Data Designer Content.....	607
Typical Use of Content Types.....	607
Integration with Web Application Servers and Languages	608
Selecting an External File Link.....	612
Indexing the Fields of a Data Design Document.....	613
Setting an Index Flag on Selected Fields	613
The Index Information File.....	614
Data Types	614
Customizing Validation Options	615
Validation Elements in the Configuration Data	615
Validation Attributes.....	616
Validation Sub-elements	617
Defining Custom Validation	621
Saving Invalid Documents	625
Calculated Fields	626
Icons on the Select Field or Group Screen.....	626
Learn More about XPath	628
Methods, Properties, and Events that Provide Access to Data Designer Fields	632
Letting Users Execute a Custom Function on a Data Design Screen..	634
The Sample Function	634
Defining the Button	635
Specifying Items in a Select List Field.....	636
Advantages of Storing List Items Externally	637
Specifying an External List	638
New API Methods in XML Data Object.....	640
Examples of Creating Select Lists.....	640
Original XML Functionality.....	644
Entering XML Content.....	644

Enabling Entry of XML Content.....	644
Retrieving XML Content	645
Validating XML Content.....	645
Saving XML Content as a Whole XML Document.....	646
How the Editor Stores and Retrieves XML Content	646
Sample Files	648
Where to Define the Appearance of XML Tags.....	648
Defining Tags in Config XML	648
Assigning XML Data in a Script.....	649
Determining How a Tag Displays	650
Tag and Data Sections	651
Tag Definition Short Cut.....	652
Inserting Custom Tags	652
How Users Insert Custom Element Tags.....	652
How Developers Insert Custom Tags	653
Troubleshooting Problems with eWebEditPro+XML ...	655
No Formatting of XML Tags	655
Attributes Do Not Display and Are Not Offered to User	656
Working with Attributes	659
Displaying Attributes	659
Setting Default Values for Attributes	660
Specifying a Number of Attribute Occurrences	660
Requiring an Attribute Value	660
How Users Insert Attribute Tags	660
Validating XML Content.....	663
Validation Files	663
Schema Files	663
XML File	663
Validating with Schemas	664
Validating Upon Tag Insertion.....	664
Validating Outside the Editor	664
Working with Schemas	666
Loading Schemas	666
Selecting Schemas	668
Validating with Schemas	669
Extracting Element Information.....	669
Externally Editing XML Tags.....	672
Retrieving the Current XML Tag.....	672
Changing an XML Tag	672
Retrieving and Setting Attributes	673
Delimiting and Un-Delimiting Attributes	673
Retrieving Attribute Values.....	674
Determining if a Tag is Selected	674
Example Usage: Reading an Attribute	674
Example Usage: Writing an Attribute.....	674
Modifying a Tag's Contents	675
Retrieving Tag Contents	675

Modifying Tag Contents	675
Implementing a Double Click Notification	676
Implementing Double Click Action.....	676
Defining External Tag Functionality	676
Script Activation of Double Click Command.....	676
Modifying the Context Menu.....	677
The Custom XML Tag DTD and Schema.....	678
The XML DTD and Schema.....	678
Custom Tag Elements	680
Alphabetical Table of Custom Tag Elements	681
CustomTag Element.....	683
Element Hierarchy	683
Child Elements	683
Attributes	683
Tagdefinitions Element.....	683
Element Hierarchy	683
Child Elements	683
Attributes	683
Tagspec Element	684
Element Hierarchy	684
Child Elements	684
Attributes	684
Types of XML Tags	686
Blocking Tags	687
Non-blocking Tags	687
Vertical and Horizontal Tags	688
Nonempty Tags.....	688
Empty Tags.....	688
Sample Vertical Tag.....	689
Sample Horizontal Tag	690
Sample Nonempty Tag	690
Sample Nonempty Tag (glyph)	690
Sample Empty Tag	691
Sample Empty Tag (glyph)	691
Caption Element	692
Element Hierarchy	692
Attributes	692
Glyph Element	693
Element Hierarchy	693
Attributes	693
Tagdefault Element	693
System Default Values	694
Element Hierarchy	694
Child Elements	694
Attributes	694
Simtaglist Element	694
How the simtaglist and simtag Elements Work.....	694
Element Hierarchy	694

Child Elements	695
Attributes	695
Simtag Element.....	695
Element Hierarchy	695
Child Elements	696
Attributes	696
Docxml Element.....	696
Element Hierarchy	696
Child Elements	696
Attributes	697
Transform Element.....	698
Element Hierarchy	699
Attributes	699
Loadsch Element	699
Element Hierarchy	699
Child Elements	700
Attributes	700
XSD Element	700
Element Hierarchy	700
Attributes	700
Elements that Control Dialog Boxes	701
taginsdlg	701
tagattrdlg.....	702
tagpropdlg.....	703
XML Commands, Methods and Parameters	705
Custom Tag Commands	705
cmdcusttagprop	705
cmdcusttaginsert	705
cmdcusttagattrs	707
cmdcustapplytag.....	707
cmdcusthidetags.....	707
cmdcustshow	707
cmdcusthide	708
Custom Tag Methods	708
Parameter Properties.....	708
xmllInfo	708
XML Additions to Editor API.....	708
Property: XmllInfo	709
Method: XMLProcessor() As XML Object.....	709
Modified API.....	709
Method getDocument().....	709
Method getHeadHTML	709
Method setDocument	710
Method setHeadHTML(strNewHead As String).....	710
Implementing a Command that Inserts a Comment.....	710
The mycomment Button	711
Determining What Happens When the Button is Pressed.....	711
Modifying the Event Handler	712

Using the Comment Sample.....	712
Custom XML Dialog Boxes	714
The Insert Custom Tags Dialog.....	714
The User Interface of the Insert Custom Tags Dialog.....	714
The Custom Tag Attributes Dialog	715
The User Interface of the Custom Tag Attributes Dialog	716
The Custom Tag Properties Dialog	716
The User Interface of the Custom Tag Properties Dialog	717
XML Objects	719
XML Object Interface	719
XML Data	719
Best Practices for Using Custom Tags.....	721
Ways to Assign Tag Definitions.....	721
Not Defining Any Tags	721
Defining in the Configuration XML Data	722
User Selectable Tag Looks	722
Defining at Load Time	723
Assigning External XML Stream.....	723
Ideas for Appearance.....	723
Use of Color.....	723
Data Width.....	724
Browser Limitations.....	724
Tag Type Considerations	725
Look.....	725
Blocking vs. Non-Blocking	725
Non-Empty Tag Problem	725
Determining Content Size.....	725
Use Tag Properties Dialog as Diagnostic Tool.....	725
Don't Use a Namespace with HTML Tags	726
Validating XHTML Content	727
Validating Content For Compliance with Accessibility Standards	727
Enabling Accessibility	727
Defining the XSLT or Web Site	728
Dialogs that Allow Input of Accessibility Information.....	731
Validating Content for Compliance with XHTML Standards	732
The Schema	732
The Web Site	733
Integrating eWebEditPro+XML.....	734
 Integrating eWebEditPro+XML with JSP	735
Using the Sample Pages	735
Creating Your Own Page	735
Including a Reference to ewebeditpro.jsp.....	735
Setting Up a Form.....	735
Placing the Editor on the Form	736

Changing Parameter Values.....	736
Inserting the Editor	737
Retrieving and Loading Data Designer Content	737
Adding a Submit Button	738
Integrating eWebEditPro+XML with ASP.....	739
Using the Sample Pages	739
Creating Your Own Page	739
Including a Reference to ewebeditpro.asp.....	739
Entering a Relative Path.....	739
Entering an Absolute Path.....	739
Setting Up a Form.....	740
Placing the Editor on the Form	740
Changing Parameter Values.....	741
Inserting the Editor as a Box	741
Retrieving and Loading Data Designer Content	742
Inserting the Editor as a Button	742
Adding a Submit Button	743
Integrating eWebEditPro+XML with ASP.NET	744
Using the Sample Pages	744
Integrating eWebEditPro+XML on an ASP.NET Page.....	744
Using a Function.....	745
Using a Custom User Control.....	746
Using a Custom Server Control.....	748
Declaring the Schema File.....	755
Integrating eWebEditPro+XML with ColdFusion	756
Creating Your Own Page	756
Setting Up a Form.....	756
Calling the eWebEditPro+XML Custom Tag.....	756
First Time Installation of eWebEditPro+XML.....	756
Adding a Submit Button	757
eWebEditPro+XML's Custom Tag.....	758
Custom Tag Attributes	758
Integrating eWebEditPro+XML with PHP.....	762
Using the Sample Pages	762
Creating Your Own Page	762
Including a Reference to ewebeditpro.php	762
Setting Up a Form.....	763
Placing the Editor on the Form	763
Changing Parameter Values.....	764
Inserting the Editor	764
Retrieving and Loading Data Designer Content	765
Adding a Submit Button	765
Integrating eWebEditPro+XML Using JavaScript	766
Using the Sample Pages	766
Formats for Placing the Editor on the Page	766
Creating Your Own Page	768
Create an HTML Page with Header and Body Tags.....	768

Include the eWebEditPro+XML JavaScript File	768
Enter a Form Element	768
Changing Parameter Values.....	768
Inserting the Editor as a Box	769
Inserting the Editor as a Button	771
Encoding Characters in the Value Attribute	771
Loading the Content.....	773
Detecting the Load Method	773
Manually Loading Content into the Editor.....	774
Saving the Content	774
Detecting when the Save Method is Invoked	774
Terminating the Save Method.....	775
Saving Content Manually.....	775
Closing a Window without Saving Content.....	775
Prevent Detecting the onsubmit Event	775
Prevent Detecting the onbeforeunload/onunload Event	775
Preventing the Save Caused by an onbeforeunload Event.....	775
Saving from One Instance of the Editor.....	776
Detecting When the Popup Editor is Activated.....	777
Testing the Page	777
Appendices	778
Appendix A: Naming the eWebEditPro+XML Editor	778
Appendix B: Error Messages	779
Appendix C: eWebEditPro+XML Architecture	785
Appendix D: Automatic Upload File Types	787
Images	788
Audio.....	790
Video.....	791
Text	792
Application (file for a specific application)	793
Other	800

Introduction

This documentation provides Web developers with information they need to deploy and customize **eWebEditPro+XML**. It explains how to perform common tasks, such as removing a button from the toolbar and creating a custom command.

IMPORTANT!

Two key sections that you *must* read are "eWebEditPro+XML Object Model" on page 2 and "eWebEditPro+XML API Cheat Sheet" on page 29. They describe how to customize **eWebEditPro+XML** using the API.

The documentation also describes how to work with the following files.

- JavaScript files
- the JavaScript objects
- the ActiveX control
- the localization files
- the configuration file
- ewebeditpropopup.htm

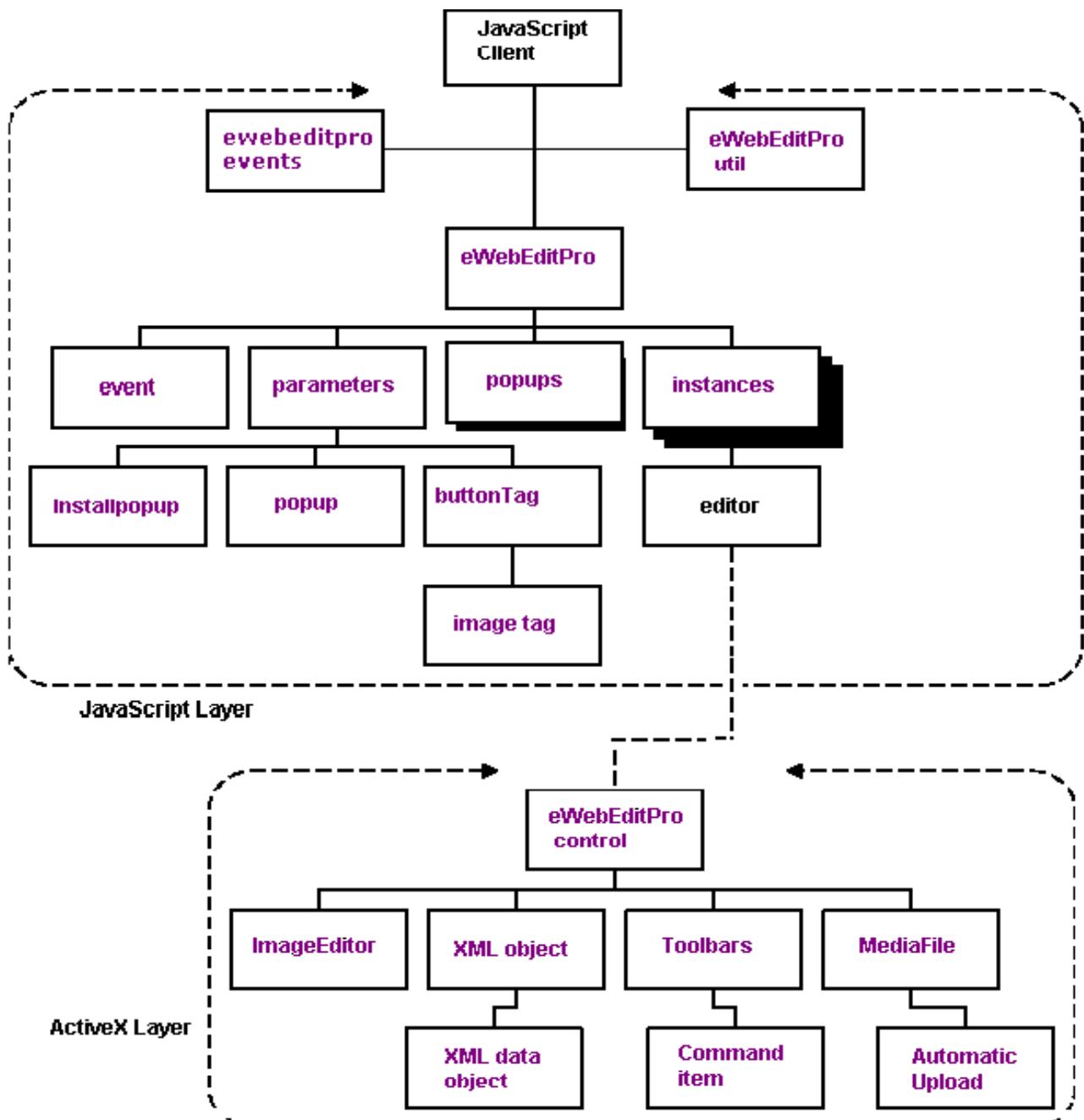
After you install **eWebEditPro+XML**, these files reside in your ewebeditpro5 directory.

NOTE

Typically, you would not change the JavaScript files. Instead, you would create a new file, define a set of functions, and include this file in the HTML. You could also define the functions directly in the HTML file.

Finally, the documentation explains other topics such as the image upload feature, style sheets, encoding special characters, and how to integrate **eWebEditPro+XML** onto a Web page using JavaScript.

eWebEditPro+XML Object Model



NOTE Methods, properties and events in the XML objects and XML Data object are only available with eWebEditPro+XML.

ewebeditproevents Object

Description: Lists eWebEditPro+XML events.

Hierarchy Location:

JavaScript Client
+-->ewebeditproevents

Child Objects: none

Syntax for retrieving object:

ewebeditpro.event

For example:

```
eWebEditPro.onready = "initTransferMethod(eWebEditPro.event.srcName)";
```

For more information see: "Event Handler Functions" on page 300; "Double-Click Element Handlers" on page 301

Name	API Type	Return Type	Description	Details
eWebEditProReady	event		Indicates it is safe to send commands to or access the MediaFile Object.	191
eWebEditProExecCommand	event		JavaScript that is called after an internal command is executed, or when an external command should be executed.	191
eWebEditProMediaSelection	event		Lets you add a media file handler.	192
eWebEditProMediaNotification	event			
eWebEditProDblClickElement	event		Occurs when a user double-clicks a hyperlink, applet, object, image or table within the editor, unless a specific event handler for hyperlink, image or table is defined.	193
eWebEditProDblClickHyperlink	event		Occurs when user double-clicks a hyperlink.	193
eWebEditProDblClickImage	event		Occurs when user double-clicks an image.	194
eWebEditProDblClickTable	event		Occurs when user double-clicks a table.	194

eWebEditProUtil Object

Description: Offers utility functions.

Hierarchy Location:

JavaScript Client

+-->eWebEditProUtil JavaScript

Child Objects: none

Syntax for retrieving object:

eWebEditProUtil;

For more information see: “eWebEditProUtil JavaScript Object” on page 307

Name	API Type	Return Type	Description	Details
editorName	property		Holds the name of the editor that opened the popup.	181
getOpenerInstance	method		Returns a reference to Instance JavaScript object responsible for opening the popup.	90
HTMLEncode	method		HTML encodes the given string.	94
isOpenerAvailable	method		Determines if page that opened the popup is still open.	99
languageCode	property		The language code of the browser.	181
queryArgs	property		The array of URL query string parameters passed to the page.	181

eWebEditPro Object

Description: Lets you add custom properties dynamically at run-time.

Hierarchy Location:

JavaScript Client

+--> eWebEditPro JavaScript Object

Child Objects: event, parameters, instances, popups

Syntax for retrieving object:

eWebEditPro;

For more information see: “JavaScript Objects” on page 300

Name	API Type	Return Type	Description	Details
{editor name}	property		A reference to an instance of the eWebEditPro+XML ActiveX control.	177

Name	API Type	Return Type	Description	Details
actionOnUnload	property		Determines how content is saved when Web page is unloaded.	177
addEventHandler	method		Defines event handlers for eWebEditPro+XML events, such as onready.	50
autoInstallExpected	method		Indicates if an automatic download and installation of eWebEditPro+XML is expected.	59
create	method		Creates an instance of an in-line editor in the page.	66
createButton	method		Creates an instance of a button which, if clicked, opens a popup window with the editor in it.	67
defineField	method		Loads more than one content field into the editor.	607
edit	method		Opens a popup window with the editor in it.	71
EstimateContentSize	method	long	Estimates the size of current content.	79
installPopup	property	boolean	If true, a window with the intro.htm page pops up.	178
instances collection	property		An array of in-line editor objects of type eWebEditProEditor or eWebEditProAlt.	178
isAutoInstallSupported	property	boolean	If true, eWebEditPro+XML can be automatically installed.	178
isChanged	method	boolean	Determines if editor content has changed.	95
isEditor	method	boolean	Indicates if an instance of an editor exists by the given name, and if the instance has a valid 'editor' property.	96
isInstalled	property	boolean	If true, eWebEditPro+XML is installed.	179
isSupported	property	boolean	If true, eWebEditPro+XML is supported in this environment. It may not be installed yet.	179
load	method		Loads content into all in-line editors on page from standard HTML elements with the same name.	105
onbeforeedit	event		Occurs when the onbeforeedit method is invoked.	188
onbeforeload	event		Occurs when the load method is invoked.	189
onbeforesave	event		Occurs when the save method is invoked.	189
oncreate	event		Occurs when the create method is invoked.	187
oncreatebutton	event		Occurs when the createButton method is invoked.	188
onedit	event		Occurs after the popup window closes.	188
onerror	event		Occurs when an error occurs because the save method failed.	190
onload	event		Occurs when the load method is complete.	190
onready	event		Occurs when it is safe to send commands or access the Media File Object.	190

Name	API Type	Return Type	Description	Details
onsave	event		Occurs when the save method is complete.	189
ontoolbarreset	event		Occurs when the editor's toolbar is initialized or reset.	189
openDialog	method		Opens the popup Web page specified by fileName.	108
parametersobject	property		An object of type eWebEditProParameters containing the default set of parameters used when creating an instance of the editor or button.	179
refreshStatus	method		Updates the value of several properties such as status, isIE, and isNetscape,	113
resolvePath	method		Prepends the URL with the eWebEditPro+XML path.	116
save	method		Saves content into all in-line editors on page from standard HTML elements with the same name.	117
status	property		Reflects the current state of eWebEditPro+XML .	179
upgradeNeeded	property	boolean	If true, an older version eWebEditPro+XML is installed and needs to be upgraded.	180
versionInstalled	Property		Retrieves the version of the control.	163

Event Object

Description: The eWebEditPro.event object is available during an event. Its properties are determined by the event.

Hierarchy Location:

`eWebEditPro JavaScript Object`

`+-->event`

Child Objects: none

Syntax for retrieving object:

`eWebEditPro.event;`

For more information see: “JavaScript Objects” on page 300

Name	API Type	Return Type	Description	Details
onbeforeedit	event		Occurs when the user clicks the button created by the createButton method.	188
onbeforeload	event		Occurs when the load method is invoked.	189
onbeforesave	event		Occurs when the save method is invoked.	189
oncreate	event		Occurs when the create method is invoked.	187
oncreatebutton	event		Occurs when the createButton method is invoked.	188

Name	API Type	Return Type	Description	Details
onedit	event		Occurs after the popup window closes.	188
onerror	event		Occurs when an error occurs because the save method failed.	190
onload	event		Occurs when the load method is complete.	190
onready	event		Occurs when it is safe to send commands or access the Media File Object.	190
onsave	event		Occurs when the save method is complete.	189
ontoolbarreset	event		Occurs when the toolbar is initialized or reset.	189
srcName	property		The name of the instance of the editor that is the source of the current event.	167
type	property		The name of the current event without the "on" prefix.	167

Parameters Object

Description: The eWebEditPro.event object is available during an event. Its properties are determined by the event.

Hierarchy Location:

eWebEditPro JavaScript Object

+-->parameters

Child Objects: installpopup, popup, Button Tag

Syntax for retrieving object:

eWebEditPro.parameters

For more information, see: "The Parameters Object" on page 306

Name	API Type	Return Type	Description	Details
buttonTag	property		Object consisting of • eWebEditProDefaults.buttonTagStart • eWebEditProDefaults.buttonTagEnd • eWebEditProMessages.popupButtonCaption <i>See Also:</i> "Button Tag Object"	167
cols	property		The number of columns in the TEXTAREA element if eWebEditPro+XML is not installed or not supported.	168
editorGetMethod	property		Lets you save either the body only or the entire HTML document from the editor.	182
embedAttributes	property		Optional attributes to the EMBED tag.	168
locale	method		Specifies the locale file to use.	106

Name	API Type	Return Type	Description	Details
maxContentSize	property		The largest number of characters that can be saved in editor.	168
objectAttributes	property		Optional attributes to the OBJECT tag.	169
onblur	event		An event that fires when the editor loses the focus. Important! This event does not work with Netscape or Firefox.	187
ondblclickelement	event		The JavaScript event that occurs when a user double-clicks any selectable element object.	186
onexeccommand	event		The default JavaScript onexeccommand handler.	187
onfocus	event		An event that fires when the editor gains the focus. Important! This event does not work with Netscape or Firefox.	187
path	property		The path to the eWebEditPro+XML files relative to the hostname.	169
preferredType	property		Specifies the type of editor to create.	169
readOnly	property		Prevents the user from modifying the editor content.	170
relocate	method		Relocates the 'on' event handlers to point to the frame where the functions are defined.	113
reset	method		Reinitializes all values to the default defined in eWebEditProDefaults (ewebeditprodefaults.js).	116
rows	property		The number of rows in the TEXTAREA element if eWebEditPro+XML is not installed or not supported.	170
textareaAttributes	property		Optional attributes to the TEXTAREA tag.	170

Popups Object

Description: An array of objects that tracks the number of "popup" editors. A popup editor is created when the createButton method is called and when the "section" editor is created.

The array can be used to determine if any popup windows are open.

Hierarchy Location:

eWebEditPro JavaScript Object

+-->popups

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.popups[ sPopupName ] ;
```

Name	API Type	Return Type	Description	Details
query	property		A query to pass parameters to the popup window.	173
url	property		The URL to the Web page that contains the editor that appears in the popup window.	173
windowFeatures	property		The parameters passed to the standard JavaScript window.open() method.	173
windowName	property		The name assigned to the popup window created by the standard JavaScript function window.open().	174
isOpen	method		Can count the number of open popup windows.	98

Instances Object

Description: Methods, properties and events that function as they do with the **eWebEditPro+XML** object but only apply to this instance of the editor.

Hierarchy Location:

eWebEditPro JavaScript Object

+-->instances

Child Objects: **eWebEditPro ActiveX Control Object**

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName];
```

For more information see: “The Instance Object” on page 304

Name	API Type	Return Type	Description	Details
addEventHandler	method		Defines event handlers for eWebEditPro+XML events, such as onready.	50
editor	property		A reference to the eWebEditPro+XML ActiveX control.	174
elemName	property		The name of the field element that contains the editor content.	174
formName	property		The name or index of the form that contains this instance of the editor.	174
height	property		The height of the editor assigned when created.	175
html	property		A string containing the HTML.	175
id	property		The name of the eWebEditPro+XML editor element in the object (Internet Explorer) or embed (Netscape) tag.	175
insertMediaFile	method		Inserts an image file (or other media file) to the editor.	94

Name	API Type	Return Type	Description	Details
isChanged	method	boolean	Returns true if content in any editor on the page was modified.	95
isEditor	method	boolean	Returns true if the .editor object is available.	97
load	method		Loads content into editor.	104
maxContentSize	property		The largest number of characters that can be saved in the editor window.	175
name	property		The name assigned to this instance of the editor when it was created.	176
onbeforeload	event		Occurs when the load method is invoked.	189
onbeforesave	event		Occurs when the save method is invoked.	189
onerror	event		Occurs when an error occurs because the save method failed. <i>See Also:</i> "The onerror Event"	190
onload	event		Occurs when the load method is complete.	190
onsave	event		Occurs when the save method is complete.	189
readOnly	property		Prevents user from modifying editor content.	176
receivedEvent	property	boolean	"True" if an event has been received from ActiveX control.	176
save	method		Saves content.	116
status	property		The status of this editor.	176
type	property		Indicates which type of editor was created on page.	176
width	property		The width of editor assigned when created.	177

InstallPopup Object

Description: This set of defaults determines the attributes of the instance of eWebEditPro+XML that appears as a popup window.

Hierarchy Location:

```
eWebEditPro JavaScript Object
    +-->parameters
        +-->installPopup
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.parameters.installPopup;
```

For more information see: "Parameters Object" on page 7

Name	API Type	Return Type	Description	Details
close	method		Closes popup window.	
open	method		Displays page specified by the installPopup parameters in popup window.	
popup	property		Lets you pass four parameters to popup Web page.	171
query	property		An optional parameter that specifies query string values to pass to page specified by URL parameter.	173
url	property		Specifies URL of Web page to display in popup window when an automatic installation is expected.	172
windowFeatures	property		Specifies popup window features as defined for standard JavaScript window.open() method.	172
windowName	property		Specifies the name of the popup window.	172

Popup Object

Description: These defaults determine how the popup window is launched.

Hierarchy Location:

```
eWebEditPro JavaScript Object
    +-->parameters
        +-->popup
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.parameters.popup;
```

For more information see: “Parameters Object” on page 7

Name	API Type	Return Type	Description	Details
query	property		A query to pass parameters to the popup window.	173
url	property		The URL to the Web page that contains the editor that appears in the popup window.	173
windowFeatures	property		The parameters passed to the standard JavaScript window.open() method.	173
windowName	property		The name assigned to the popup window created by the standard JavaScript function window.open().	174

Button Tag Object

Description: Lets you determine the form of the popup edit button.

Hierarchy Location:

```
eWebEditPro JavaScript Object  
+-->parameters  
    +-->buttonTag
```

Child Objects: image tag

Syntax for retrieving object:

```
eWebEditPro.parameters.buttonTag;
```

For more information see: “Customizing the createButton Command” on page 254

Name	API Type	Return Type	Description	Details
End	property		Determines the end of the HTML that appears on the popup edit button.	165
Start	property		Determines the beginning of the HTML that appears on the popup edit button.	165
tagAttributes	property		Used to assign custom attributes to the popup edit button.	166
type	property		Determines the form of the popup edit button.	166
value	property		Determines the value of the popup edit button.	166

Image Tag Object

Description: Lets you customize the image that appears on the popup edit button.

Hierarchy Location:

```
eWebEditPro JavaScript Object  
+-->parameters  
    +-->buttonTag  
        +-->imageTag
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.parameters.buttonTag.imageTag;
```

For more information see: “Customizing the createButton Command” on page 254

Name	API Type	Return Type	Description	Details
alt	property		Determines the alt text that appears on the popup edit button.	165
border	property		Determines the size of the border on the popup edit button.	165
height	property		Determines the height of the popup edit button.	165
src	property		Determines the source of the image that appears on the on the popup edit button.	165
width	property		Determines the width of the popup edit button.	165

eWebEditPro ActiveX Control Object

Description: Lets you control eWebEditPro+XML's functionality and content

Hierarchy Location:

eWebEditPro JavaScript Object

+-->instances

+-->eWebEditPro ActiveX Control

Child Objects: Image editor, XML Object, Toolbars, Media File

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor;
```

For more information see: "ActiveX Control" on page 309

Name	API Type	Return Type	Description	Details
addInlineStyle	Style Sheet Method	string	Adds an inline <STYLE>... </STYLE> to document header.	51
addLinkedStyleSheet	Style Sheet Method	string	Adds linked style sheet reference to document header.	52
BaseUrl	Property	string	This property sets the current URL of the editor to the specified location.	166
bodyStyle	Property		Cascading style sheet (CSS) attribute values.	159
BodyStyle	Style Sheet Method	string	Sets/gets the document's body style.	61
CharSet	Property		The charset value for a page.	160
ClearStylesFromTags	Style Sheet Method		Removes style attribute from all tags in document.	63
Config	Property		The URL of the config XML file.	160

Name	API Type	Return Type	Description	Details
disableAllStyleSheets	Style Sheet Method		Enables or disables all style sheets for an editor.	69
Disabled	Property	boolean	When set to true, the editor is disabled.	160
disableStyleSheet	Style Sheet Method		Enables or disables linked or inline style sheet as identified by its title.	69
ExecCommand	Method		Causes the editor to perform the specified operation.	81
Focus	Method		Programmatically sets focus to eWebEditPro+XML editor using JavaScript.	82
Get WDDX	Property	string	Sets or retrieves assigned WDDX data.	162
GetActiveStyleSheetTitles	Style Sheet Method	string	Returns a comma-delimited list of the titles of active styles.	83
getBodyHTML	Method		Saves content within the BODY tags as HTML.	83
getBodyText	Method		Returns content text without formatting.	84
GetContent	Method	string	Retrieves specified content type from current edit session.	85
getDocument	Method		Saves entire HTML document currently in editor.	85
getHeadHTML	Method		Returns <HEAD> through </HEAD> HTML of current document as a string, including the HEAD tags.	88
getProperty	Method		Reads from ActiveX control property.	90
getPropertyBoolean	Method		Returns value of a Boolean property.	90
getPropertyInteger	Method		Returns value of a Numeric property.	91
getPropertyString	Method		Returns value of a String property.	91
getSelectedHTML	Method	string	Returns currently selected content including any HTML tags.	92
getSelectedText	Method	string	Returns currently selected text with no formatting.	92
hideAboutButton	Property	boolean	Can remove the About button from the toolbar.	162
IsDirty	Property	boolean	Returns "true" if content has changed.	162
isEditorReady	Method	boolean	If "true", editor is ready to process a command.	97
IsTagApplied	Method	boolean	Indicates if a specified XML tag can be applied at the current cursor location.	99
License	Property		The license keys of the editor.	162
Locale	Property		The URL of the localization directory or file.	162
MediaFile	Method	media file object	Returns reference to the Media File object.	106

Name	API Type	Return Type	Description	Details
onblur	event		An event that fires when the editor loses the focus.	187
ondblclickelement	event		Double-clicking on a hyperlink, applet, object, image, or table causes this event to fire.	186
onexeccommand	event		Raised after a toolbar button is pressed, a toolbar dropdown list item is selected, or a context menu (right-click menu) item is selected.	186
onfocus()	event		An event that fires when the editor gains the focus.	187
pasteHTML	Method		Replaces selected content with string passed to pasteHTML.	109
pasteText	Method		Replaces selected content with string passed to pasteText.	110
PopulateTagsWithStyles	Style Sheet Method	boolean	Applies current, active styles to content's tags.	110
ReadOnly	Property	boolean	Prevents user from modifying editor content.	162
SetContent	Method	string	Assigns given content to the editor session.	119
setDocument	Method		Replaces entire document with specified document.	120
setHeadHTML	Method		Sets <HEAD> through </HEAD> portion of the document header.	119
setProperty	Method		Writes to ActiveX control property.	124
ShowActiveStylesDetails	Style Sheet Method	string	Returns a comma-delimited list of the active style sheet titles and style information	126
SrcPath	Property	string	Specifies where eWebEditPro+XML is installed.	163
StyleSheet	Property	string	Specifies style sheet file (CSS) to apply to editor content.	163
TagCount	Method	long	Indicates how many times a specified XML tag exists in the content.	127
Title	Property		A document title for page.	163
Toolbars	Method	Toolbar Control Object	Returns a reference to the Toolbar Interface object.	130
version	property		The version of the control.	180
xmlInfo	Property		Dynamically assigns XML and custom tag configuration data that is external to normal configuration data.	164
XMLProcessor	Method	XML Object	Retrieves interface to XML Object.	133

Image Editor Object

Description: All all methods available to manipulate WebImageFX, such as displaying the dialog that lets the user save the file.

Hierarchy Location:

```
eWebEditPro JavaScript Object  
+-->instances  
    +-->eWebEditPro ActiveX Control  
        +-->ImageEditor
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.ImageEditor();
```

For more information see: “WebImageFX” on page 577

Name	API Type	Return Type	Description	Details
AskOpenFile	Method	boolean	Displays a dialog that prompts the user to select an image to edit.	57
AskSaveAs	Method	string	Displays a dialog that asks user to select a format and file name for current image.	57
AskSelectColor	Method	none	Displays a dialog for user to choose color and line size of recently-drawn annotation.	57
ConvertImage	Method	string	Converts specified image into file format requested by client.	65
CreateNew	Method	boolean	Creates or saves a new image.	67
EditCommandComplete	Event	n/a	Notifies client application or script that user edit command has completed.	183
EditCommandStart	Event	n/a	Notifies client application or script that user edit command has started.	183
EditComplete	Event	n/a	Notifies client application or script that editing session has completed.	184
EditFile	Method	boolean	Loads the given file for user editing.	71
EditFromHtml	Method	string	Parses specified HTML tag and extracts information about image and associated named data from attributes.	72
EnableCreation	Method	boolean	Enables or disables user interface that allows user to create new image.	76
EnableFormatChange	Method	boolean	Enables or disables user's ability to change the file format and select the number of colors for image.	76

Name	API Type	Return Type	Description	Details
EnableNameChange	Method	boolean	Enables or disables user's ability to change the name of image file.	77
ErrorClear	Method	void	Clears any current errors.	78
ErrorDescription	Method	string	Retrieves a text description of the last error encountered.	78
ErrorValue	Method	long	Returns a numeric value representing the last error encountered.	79
ExecCommand	Method	none	Directly executes a command name with parameters, without going through eWebEditPro+XML 's command mechanism.	80
GetImageInformation	Method	string	Retrieves specified information about an image.	89
GetValidFormats	Method	string	Retrieves current set of valid file formats supported by feature.	92
ImageEditor	Method		Retrieves Image Edit object that exists within WebImageFX.	94
ImageError	Event	n/a	Notifies client application or script that error has occurred.	184
IsDirty	Method	boolean	Returns a non-zero (boolean true) value if user modified image.	96
IsPresent	Method	boolean	Returns true if WebImageFX is installed properly on client system.	99
isVisible	Method	boolean	This method returns true if WebImageFX is visible to user from within eWebEditPro+XML .	101
LoadedFileName	Method	string	Returns name of loaded image file.	105
LoadingImage	Event	n/a	Notifies client application or script that image file has loaded.	185
PublishHTML	Method	string	Formats named values into HTML tag that contains attribute/value combinations.	111
Save	Method	string	Saves currently edited image with currently selected file parameters.	116
SaveAs	Method	string	Saves the currently edited image with the specified parameters.	117
SavedFileName	Method	string	Returns name that file was actually saved as.	117
SavingImage	Event	n/a	Called before current image is saved to local file system.	185
SetConfig	Method	string	Specifies which configuration file to use for controlling WebImageFX.	119
SetLocale	Method	string	Specifies a Locale translation file to use.	123

Name	API Type	Return Type	Description	Details
SetValidFormats	Method	long	Specifies a set of formats that are considered valid by a client application or script.	125
Thumbnail	Method	string	Creates a thumbnail of the current image or a specified image file.	127

XML Object

Description: Lets you control all aspects of the XML object, such as retrieving the namespace of the current schema, and retrieving or setting the XML document's root tag.

Hierarchy Location:

eWebEditPro JavaScript Object

+-->instances

+-->eWebEditPro ActiveX Control

+-->XML object

Child Objects: XML Data object

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.XMLProcessor();
```

For more information see: "XML Object Interface" on page 719

Name	API Type	Return Type	Description	Details
ActiveSchema	property	string	Selects or retrieves the namespace of the current schema.	136
ActiveTag	method	string	Retrieves a reference to a CXMLElement object that contains the tag applied to the current selection.	50
AddSchema	method	boolean	Loads a schema file.	54
AnySchemasLoaded	method	boolean	Returns True if any schemas are loaded.	55
ApplyTag	method	boolean	Applies CXMLElement object information to the current inner tag in the content of the object.	56
AutoFillIn	property	boolean	Determines if an element is automatically filled in when inserted into the editor.	136
AvailableSchemas	method	string	Retrieves namespaces of all loaded schemas.	60
BuildErrorDescEng	method	string	Retrieves a full error description in English.	61
Clicktag	event		Occurs when user clicks a Data Design field	182
DocumentTemplate	method	string	Uses active schema to generate a skeleton document template.	70

Name	API Type	Return Type	Description	Details
DuplicateTag	method	string	Creates a duplicate of a given tag.	70
ErrorCode	property	long	Contains the error code for the last error.	137
ErrorFilePos	property	long	If error, position in the file where error occurred.	137
ErrorLine	property	long	If error, line where error occurred.	137
ErrorPos	property	long	If error, position in line where error occurred.	137
ErrorReason	property	string	If error, contains a short description why there was an error.	138
ErrorSrcText	property	string	If error, the source within the file that causes the error.	138
ErrorURL	property	string	If error, the URL of the loaded schema that contains the error.	138
FindDataField	method	string	Returns CXMLElement object.	81
getProperty	method	string	Retrieves the property name given.	90
getPropertyBoolean	method	boolean	Retrieves the property name given as a string.	91
getPropertyInteger	method	integer	Retrieves the property name given as an integer.	91
getPropertyString	method	string	Retrieves the property name given as a boolean.	91
RootTag	property	string	Retrieves or sets the root tag of a loaded XML document.	139
SchemaExists	method	boolean	Allows client to check if the schema assigned the given namespace is loaded.	118
setProperty	method	none	Sets the named property to the value given.	124
ShowRootTag	property	boolean	Determines if the user can see root tag in the header of the XML data.	139
Transform	method	string	Performs a transformation on the document content given to the method.	130
TransformOnLoad	property	string	Sets or reads the XSLT file used when whole content is loaded.	140
TransformOnSave	property	string	Sets or reads XSLT file used when whole content is extracted from the editor.	140
Validate	method	boolean	Validates a given set of content.	132
XmlHeader	property	string	An alternative, and more direct, method of accessing XML header information.	141

Toolbars Object

Description: Contains properties and methods that let you control menu, button, and command functionality.

Hierarchy Location:

```

eWebEditPro JavaScript Object
+-->instances
    +-->eWebEditPro ActiveX Control
        +-->Toolbars

```

Child Objects: CommandItem

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.Toolbars();
```

For more information see: “The Toolbar Object Interface” on page 258

Name	API Type	Return Type	Description	Details
CommandAdd	method	string	Adds a command to the specified toolbar.	64
CommandDelete	method	none	Deletes a command from a toolbar.	65
CommandItem	method	string	Retrieves the interface directly to the command item.	65
HideAbout	method	boolean	Hides the about command button.	93
HideAllMenus	method	none	Hides all toolbar menus.	93
PopupMenu	method	none	Brings up a popup menu.	111
SeparatorBarAdd	method	boolean	Adds a separator bar to the specified toolbar.	118
SeparatorSpaceAdd	method	boolean	Adds a separator space to the specified toolbar.	118
ShowAbout	method	boolean	Shows the about button	126
ShowAllMenus	method	none	Restores the view of menus hidden with HideAllMenus.	127
ToolbarAdd	method	etb ErrorValues value	Creates a toolbar and adds it to the toolbars available to the user.	129
ToolbarModify	method	etb ErrorValues value	Modifies an existing toolbar.	129

Media File Object

Description: Contains information about the uploaded file, such as the source location, the destination and size of the image.

Hierarchy Location:

```

eWebEditPro JavaScript Object
+-->instances
    +-->eWebEditPro ActiveX Control
        +-->Media File

```

Child Objects: Automatic Upload**Syntax for retrieving object:**

```
eWebEditPro.instances[sEditorName].editor.MediaFile();
```

For more information see: “[Managing Images](#)” on page 455 and
“[Programmatically Accessing Media File Properties](#)” on page 488

Name	API Type	Return Type	Description	Details
Alignment	property	string	Determines image's alignment on the page.	149
AllowSubDirectories	property	boolean	Determines if user can select subdirectories.	149
AllowUpload	property	boolean	Determines if user can upload files from local PC to server.	148
BaseURL	property	string	The base URL value set in the editor.	150
BorderSize	property	integer	The size of the image's border in pixels.	150
DefDestinationDir	property	string	The destination path where the image will be placed.	150
DefSourceDir	property	string	The initial directory that appears when the user is selecting a local file.	150
Domain	property	string	The domain name of the upload server.	151
FileExistsLocally	method	boolean	Determines if file exists on local system.	81
FileSize	property	long	The size of the image file in bytes.	151
FileTitle	property	string	The title of the file.	151
FileType	property	string	The type of file.	151
FWLoginName	property	string	User's login name for the firewall. Not currently used.	152
FWPassword	property	string	User's password for the firewall. Not currently used.	152
FWPort	property	integer	The firewall port to use for any transfer.	152
FWProxyServer	property	string	Firewall proxy server. Not currently used.	152
FWUse	property	boolean	If true, a firewall mechanism is used. Not currently used.	152
FWUsePassV	property	boolean	If true, PASV mode FTP is enabled.	152
Get EnablePathResolution	property	boolean	Enables path resolution functionality.	153
Get IsValid	property	boolean	Returns whether current upload connection is valid.	153
Get ShowResolutionOverride	property	boolean	If “true”, user can manually enable or disable path resolution mechanism.	153
Get XferType	property	string	Retrieves or sets the transfer type string.	153
getProperty	method	string	Retrieves the property name given.	90

Name	API Type	Return Type	Description	Details
getPropertyBoolean	method	boolean	Retrieves the property name given as a boolean.	91
getPropertyInteger	method	integer	Retrieves the property name given as an integer.	91
getPropertyString	method	string	Retrieves the property name given as a string.	91
HandledInternally	property	boolean	Determines if the upload has already been handled internally.	153
HorizontalSpacing	property	integer	Horizontal spacing attribute to use in HTML.	154
ImageHeight	property	integer	The height of the image.	154
ImageWidth	property	integer	The width of the image.	154
IsLocal	property	boolean	Set to true if a local file will be placed into the SrcFileName property.	154
LoginName	property	string	The login name of the user uploading the image.	154
MaxFileSizeK	property	integer	Maximum size in kilobytes of image to be uploaded.	155
MediaType	property	string	Determines which valid extensions are provided in the Media File Selection dialog.	155
NeedConnection	property	boolean	Determines if a connection is necessary with the current upload method.	155
Password	property	string	The password of the user uploading the image.	155
Port	property	integer	The port to use for uploads.	155
ProxyServer	property	string	The name of the proxy server to use with uploads.	156
RemotePathFileName	property	string	The remote path and name of the currently selected file.	156
ResolveMethod	property	string	The method by which the image source path is resolved.	156
ResolvePath	property	string	The path used to resolve an image path when GIVEN is the resolution method.	156
RetrieveHTMLString	method	boolean	Returns HTML string to be used for insertion into HTML.	116
setProperty	method	none	Sets the named property to the value given.	124
ShowHeight	property	integer	The height attribute of the HTML image tag.	157
ShowWidth	property	integer	The width attribute for the HTML image tag.	157
SrcFileName	property	string	The full location of the source file.	157
TransferMethod	property	string	The name of the upload method used if the ProvideMediaFile method is called.	157
TransferRoot	property	string	The destination path where the image will be placed.	158
UseHTMLString	method	string	Information from given HTML string is placed into the appropriate Media object properties.	131

Name	API Type	Return Type	Description	Details
UsePassV	property	boolean	If true, FTP works in passive mode.	158
ValidConnection	property	boolean	If true, system made valid connection with current connection parameters.	158
ValidExtensions	property	string	File extensions of images that can be uploaded.	158
VerticalSpacing	property	integer	The value of the vertical spacing attribute of the HTML image tag.	158
WebPathName	property	string	The Web accessible name of the specified file.	159
WebRoot	property	string	The base location for accessing uploaded images from a Web page.	159

XML Data Object

Description: Lets you retrieve and affect selected tags in the content. For example, you can process attribute values into a set of attributes and values separated by a delimiter.

Hierarchy Location:

```
eWebEditPro JavaScript Object
+-->instances
    +-->eWebEditPro ActiveX Control
        +-->XML Object
            +-->XML Data
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.XMLProcessor().XMLData();
```

For more information see: “XML Data” on page 719

Name	API Type	Return Type	Description	Details
addListItem	method		Adds an item to the end of the list.	53
AllowEdit	property	boolean	Determines if user can edit tag contents.	142
Attributes	property	string	Attributes currently used by the specific tag in the XML data.	142
AttributeValueDefault	method	string	Returns the default value defined in the schema for the given attribute in the current element.	58
CanInsert	method	boolean	Returns a value that indicates whether or not a user can insert a tag at the current position.	61
ClearList	method		Removes all the items from list.	62

Name	API Type	Return Type	Description	Details
Content	property	string	Retrieves or sets the content within the tag.	143
DataStyle	property	string	Contains the style parameters used to display the content of the tag.	143
DelimitAttributes	method	string	Helps a client script process attributes returned from Attributes property.	68
Description	property	string	The description of the tag displayed to the user.	144
ElementAttributes	method	string	Returns valid attributes to the element delimited with the value specified in the strDelim string value.	72
ElementAttributeValues	method	string	Retrieves the valid values defined in the schema for the given attribute in the current element.	73
ElementChildren	method	string	Retrieves the valid elements defined in the schema for the current element.	74
ElementIsEmpty	method	boolean	Returns true if the selected element is defined to be empty.	75
ElementMaxCount	method	long	Returns the maximum times an element can appear within its parent element.	75
ElementMinCount	method	long	Returns the minimum times an element can appear within its parent element	76
FormatDelimitedAttributes	method	string	When attributes are contained in a delimited string that uses a pipe () as a delimiter, this method reformats them as they would appear in an element.	82
getProperty	method	string	Retrieves the property name given.	90
getPropertyBoolean	method	boolean	Retrieves the property name given as a string.	91
getPropertyInteger	method	integer	Retrieves the property name given as an integer.	91
getPropertyString	method	string	Retrieves the property name given as a boolean.	91
GetTagAttribute	method	string	Retrieves the value of the given attribute within the currently selected tag.	92
Field	method	string	Returns XPath of selected Data Design field.	93
Icon	property	string	Contains the path of the icon shown with the tag description.	144
IsAttributeRequired	method	boolean	Returns true if the given attribute is defined as required in the selected element.	95
IsDataField	method	boolean	Returns true if current tag is a Data Design tag.	96
IsValid	method	boolean	Offers a quick check to determine if the tag retrieved with ActiveTag is a valid tag.	100
ListLength	method		Returns the number of items in the list.	105
LoadList	method		Creates the list of items from an XML document.	105

Name	API Type	Return Type	Description	Details
MinChildElementCount	method	long	Returns the minimum number of times a child element can exist under the current element.	107
OuterXML	method	string	Returns the XML of the custom tag as a string, for example <mytag>some text</mytag>.	109
RemoveListItem	method		Removes an item from the list.	115
setProperty	method	none	Sets the named property to the value given.	124
SetTagAttribute	method	boolean	Sets the given attribute value to the value sent to the method.	125
ShowIcon	property	boolean	Contains whether an icon is shown to the user with the tag.	144
ShowName	property	boolean	Determines if the name, or element description, is shown with the element.	144
TagName	property	string	Contains the name of the element.	145
TagStyle	property	string	Contains the style parameters used to display the tag.	145
TagType	property	long	Contains the type of flag that defines how the element is arranged when formatted.	145
ValidChildElement	method	boolean	Offers a quick check to determine if a given element is valid within the current element.	133

Command Item Object

Description: Lets you manage the user interface commands. For example, you can disable a command, modify the button caption or tooltip text, etc.

Hierarchy Location:

```
eWebEditPro JavaScript Object
    +-->instances
        +-->eWebEditPro ActiveX Control Object
            +-->toolbars
                +-->Command Item
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.Toolbars().CommandItem(sCommandName);
```

For more information see: “The Toolbar Object Interface” on page 258

Name	API Type	Return Type	Description	Details
AddlItem	method	none	In an edit control, it sets the text. In a list box, it adds an item to the dropdown list.	52
Clear	method	none	In a list box, it clears all entries. In an edit box, it clears the text. In a toggle, it ensures that it is un-toggled.	62
CmdCaption	property	string	Retrieves the caption.	134
CmdData	property	long	Sets the current item to the entry that contains the long data value associated with the text.	134
CmdFirst	method	boolean	Sets the command object to look at the first command in the menu or toolbar.	63
CmdGray	property	boolean	The command is disabled and displayed as a grayed image.	134
CmdIndex	property	integer	Sets the currently selected index and retrieves the currently selected index into the list box.	134
CmdName	property	string	Returns the command name associated with the button.	134
CmdNext	method	boolean	Sets the command object to look at the next command in the menu or toolbar.	63
CmdStyle	property	integer	Reflects the style of the command.	135
CmdText	property	string	Sets the current selection for a list box.	135
CmdToggledOn	property	boolean	Only available to Toggle style buttons. If true, the button appears pressed in or selected. If false, it appears popped out or unselected.	135
CmdToolTipText	property	string	Contains the tooltip text associated with a command.	135
CmdType	property	etbCommand Styles	The command type assigned during the creation of the command.	135
CmdVisible	property	boolean	Reflects the visibility of a command. If true, the user can see the command.	135
FirstCommand	method	boolean	Sets the current reference to the first command available.	82
getProperty	method	string	Reads from the ActiveX control property.	90
getPropertyBoolean	method	boolean	Returns the value of a Boolean property.	91
getPropertyInteger	method	integer	Returns the value of a numeric property.	91
getPropertyString	method	string	Returns the value of a String property.	91
isValid	method	boolean	Returns "true" if the interface references a valid command.	100
ListCommandName	method	string	Returns the name of the command associated with the item at the index specified.	101

Name	API Type	Return Type	Description	Details
MaxListboxWidth	property	integer	Sets or retrieves the width of an edit box or a list box in characters.	136
NextCommand	method	boolean	Sets the current reference to the next command available.	108
setProperty	method	none	Writes to the ActiveX control property.	124

Automatic Upload Object

Description: Lets you programmatically control the Automatic Upload feature. For example, you can specify the server to use with the receiving page.

Hierarchy Location:

```
eWebEditPro JavaScript Object
+-->instances
    +-->eWebEditPro ActiveX Control Object
        +-->Media File
            +-->Automatic Upload
```

Child Objects: none

Syntax for retrieving object:

```
eWebEditPro.instances[sEditorName].editor.MediaFile().AutomaticUpload();
```

For more information see: “Automatic Upload Object” on page 562

Name	API Type	Return Type	Description	Details
AddFileForUpload	method	none	Adds file to list of files to upload.	51
AddNamedData	method	boolean	Adds named data set to individual upload files in file store.	54
AllowUpload	property	boolean	Enables or disables automatic upload feature.	148
ContentDescription	property	string	Description string sent to the server when content is posted.	148
ContentTitle	property	string	The title of the content posted to the server.	148
ContentType	property	string	The type of content posted to the server.	148
GetFieldValue	method	string	Reads value from the given data item.	85
GetFileDescription	method	string	Returns description of file in list of files added for upload.	87
GetFileStatus	method	long	Retrieves upload status of file in list of files added for upload.	87
ListFilesWithStatus	method	string	Retrieves a list of files with a specified status.	101

Name	API Type	Return Type	Description	Details
LoginName	property	string	The login name of the user uploading the image.	146
LoginRequired	property	boolean	Enables or disables the act of logging into a remote site.	147
Password	property	string	The password of the user uploading the image.	147
Port	property	long	The port used for HTTP posting or FTP transfer.	149
ReadNamedData	method	string	Retrieves the data value of the data name from the file specified.	112
ReadResponseHeader	method	string	Retrieves the header of the response sent by the server.	112
ReadUploadResponse	method	string	Reads the full text returned from the server as a response to the upload.	113
RemoveFieldValue	method	none	Removes given data item so it is not sent with the upload.	114
RemoveFileForUpload	method	none	Removes a specified file from the list of files for uploading.	114
RemoveNamedData	method	boolean	Removes the named data set from the file specified.	115
ServerName	property	string	Specifies the server to use with the receiving page.	146
SetFieldValue	method	none	Assigns a data item to be sent with the file.	121
SetFileDescription	method	none	Sets description of specified file.	121
SetFileStatus	method	none	Sets status of given file.	122
TransferMethod	property	string	Specifies how the Automatic Upload mechanism performs an upload when local files are detected.	157
TransferRoot	property	string	The destination path where the image will be placed.	147
UploadConfirmMsg	method	none	Sets user message displayed on the user intervention dialog.	131
ValidExtensions	property	string	The file extensions of images that can be uploaded, entered as a comma-delimited string.	147
WebRoot	property	string	The base location for accessing uploaded images from a Web page.	159

eWebEditPro+XML API Cheat Sheet

This section first lists and briefly describes all elements in alphabetical order. After that, it provides a detailed description for each **method**, **property**, and **event**.

You can use this information to customize your implementation of **eWebEditPro+XML** via scripting.

NOTE

Methods, properties and events in the XML objects and XML Data object are only available with eWebEditPro+XML.

Alphabetical List of Methods, Properties and Events

Method/Property/ Event	Details	In object	Description
{editor name}	177	"eWebEditPro Object"	A reference to an instance of the eWebEditPro+XML ActiveX control.
actionOnUnload	177	"eWebEditPro Object"	Determines how content is saved when Web page is unloaded.
ActiveSchema	136	"XML Object"	Selects or retrieves the namespace of the current schema.
ActiveTag	50	"XML Object"	Retrieves a reference to a CXMLData object that contains the tag applied to the current selection.
addEventHandler	50	"eWebEditPro Object"	Defines event handlers for eWebEditPro+XML events, such as onready.
addEventHandler	50	"Instances Object"	Defines event handlers for eWebEditPro+XML events, such as onready.
AddFileForUpload	51	"Automatic Upload Object"	Adds file to list of files to upload.
addInlineStyle	51	"Parameters Object"	Adds an inline <STYLE>... </STYLE> to document header.
AddItem	52	"Command Item Object"	In an edit control, it sets the text. In a list box, it adds an item to the dropdown list.
addLinkedStyleSheet	52	"Parameters Object"	Adds linked style sheet reference to document header.
addListItem	54	"XML Data Object"	Adds an item to the end of the list.
AddNamedData	54	"Automatic Upload Object"	Adds named data set to individual upload files in file store.

Method/Property/ Event	Details	In object	Description
AddSchema	54	"XML Object"	Loads a schema file.
Alignment	149	"Parameters Object"	Determines image's alignment on the page.
AllowEdit	142	"XML Data Object"	Determines if user can edit tag contents.
AllowSubDirectories	149	"Parameters Object"	Determines if user can select subdirectories.
allowupload	150	"Parameters Object"	Determines if user can upload files from local PC to server.
AllowUpload	148	"Automatic Upload Object"	Enables or disables automatic upload feature.
alt	165	"Image Tag Object"	Determines the alt text that appears for a popup window.
AnySchemasLoaded	55	"XML Object"	Returns true if any schemas are loaded.
ApplyTag	56	"XML Object"	Applies CXMLData object information to the current inner tag in the content of the object.
AskOpenFile	57	"Image Editor Object"	Displays a dialog that prompts the user to select an image to edit.
AskSaveAs	57	"Image Editor Object"	Displays a dialog that asks user to select a format and file name for current image.
AskSelectColor	57	"Image Editor Object"	Displays a dialog for user to choose color and line size of recently-drawn annotation.
Attributes	142	"XML Data Object"	Attributes currently used by the specific tag in the XML data.
AttributeValueDefault	58	"XML Data Object"	Returns the default value defined in the schema for the given attribute in the current element.
AutoFillIn	136	"XML Object"	Determines if an element is automatically filled in when inserted into the editor.
autoInstallExpected	59	"eWebEditPro Object"	Indicates if an automatic download and installation of eWebEditPro+XML is expected.
AvailableSchemas	60	"XML Object"	Retrieves namespaces of all loaded schemas.
BaseUrl	166	"eWebEditPro ActiveX Control Object"	This property sets the current URL of the editor to the specified location.
BaseUrl	150	"Parameters Object"	The base URL value set in the editor.
bodyStyle	159	"eWebEditPro ActiveX Control Object"	Cascading style sheet (CSS) attribute values.

Method/Property/ Event	Details	In object	Description
BodyStyle	61	"eWebEditPro ActiveX Control Object"	Sets/getts the document's body style.
border	165	"Image Tag Object"	Determines the size of the border in a popup window.
BorderSize	150	"Parameters Object"	The size of an image's border in pixels.
BuildErrorDescEng	61	"XML Object"	Retrieves a full error description in English.
buttonTag	167	"Parameters Object"	Object consisting of <ul style="list-style-type: none"> • eWebEditProDefaults.buttonTagStart • eWebEditProDefaults.buttonTagEnd • eWebEditProMessages.popupButtonCaption <i>See Also:</i> "Button Tag Object"
CanInsert	61	"XML Data Object"	Returns a value that indicates whether or not a user can insert a tag at the current position.
CharSet	160	"eWebEditPro ActiveX Control Object"	The charset value for a page.
Clear	62	"Command Item Object"	In a list box, it clears all entries. In an edit box, it clears the text. In a toggle, it ensures that it is un-toggled.
ClearList	62	"XML Data Object"	Removes all the items from list.
ClearStylesFromTags	63	"eWebEditPro ActiveX Control Object"	Removes style attribute from all tags in document.
clicktag	182	"XML Object"	Occurs when user clicks a Data Design field.
clientInstall	168	"Parameters Object"	The directory in which the client installation file resides.
CmdCaption	134	"Command Item Object"	Retrieves the caption.
CmdData	134	"Command Item Object"	Sets the current item to the entry that contains the long data value associated with the text.
CmdFirst	63	"Command Item Object"	Sets the command object to look at the first command in the menu or toolbar.
CmdGray	134	"Command Item Object"	The command is disabled and displayed as a grayed image.
CmdIndex	134	"Command Item Object"	Sets the currently selected index and retrieves the currently selected index into the list box.
CmdName	134	"Command Item Object"	Returns the command name associated with the button.

Method/Property/ Event	Details	In object	Description
CmdNext	63	"Command Item Object"	Sets the command object to look at the next command in the menu or toolbar.
CmdStyle	135	"Command Item Object"	Reflects the style of the command.
CmdText	135	"Command Item Object"	Sets the current selection for a list box.
CmdToggledOn	135	"Command Item Object"	Only available to Toggle style buttons. If true , the button appears pressed in or selected. If false , it appears popped out or unselected.
CmdToolTipText	135	"Command Item Object"	Contains the tooltiptext associated with a command.
CmdType	135	"Command Item Object"	The command type assigned during the creation of the command.
CmdVisible	135	"Command Item Object"	Reflects the visibility of a command. If true , the user can see the command.
cols	168	"Parameters Object"	The number of columns in the TEXTAREA element if eWebEditPro+XML is not installed or not supported.
CommandAdd	64	"Toolbars Object"	Adds a command to the specified toolbar.
CommandDelete	65	"Toolbars Object"	Deletes a command from a toolbar.
CommandItem	65	"Toolbars Object"	Retrieves the interface directly to the command item.
Config	160	"eWebEditPro ActiveX Control Object"	The URL of the config XML file.
Content	143	"XML Data Object"	Retrieves or sets the content within the tag.
ContentDescription	148	"Automatic Upload Object"	Description string sent to the server when content is posted.
ContentTitle	148	"Automatic Upload Object"	The title of the content posted to the serve.
ContentType	148	"Automatic Upload Object"	The type of content posted to the server.
ConvertImage	65	"Image Editor Object"	Converts specified image into file format requested by client.
create	66	"eWebEditPro Object"	Creates an instance of an in-line editor in the page.
createButton	67	"eWebEditPro Object"	Creates an instance of a button which, if clicked, opens a popup window with the editor in it.

Method/Property/ Event	Details	In object	Description
CreateNew	67	"Image Editor Object"	Creates or saves a new image.
DataStyle	143	"XML Data Object"	Contains the style parameters used to display the content of the tag.
DefDestinationDir	150	"Parameters Object"	The destination path where the image will be placed.
DefineField	607	"eWebEditPro Object"	Loads more than one content field into the editor.
DefSourceDir	150	"Parameters Object"	The initial directory that appears when the user is selecting a local file.
DelimitAttributes	68	"XML Data Object"	Helps a client script process attributes returned from Attributes property.
Description	144	"XML Data Object"	The description of the tag displayed to the user.
disableAllStyleSheets	69	"eWebEditPro ActiveX Control Object"	Enables or disables all style sheets for an editor.
Disabled	160	"eWebEditPro ActiveX Control Object"	When set to true , the editor is disabled.
disableStyleSheet	69	"eWebEditPro ActiveX Control Object"	Enables or disables linked or inline style sheet as identified by its title.
DocumentTemplate	70	"XML Object"	Uses active schema to generate a skeleton document template.
Domain	151	"Parameters Object"	The domain name of the upload server.
DuplicateTag	70	"XML Object"	Creates a duplicate of a given tag.
edit	71	"eWebEditPro Object"	Opens a popup window with the editor in it.
EditCommandComplete	183	"Image Editor Object"	Notifies client application or script that user edit command has completed.
EditCommandStart	183	"Image Editor Object"	Notifies client application or script that user edit command has started.
EditComplete	184	"Image Editor Object"	Notifies client application or script that editing session has completed.
EditFile	71	"Image Editor Object"	Loads the given file for user editing.
EditFromHtml	72	"Image Editor Object"	Parses specified HTML tag and extracts information about image and associated named data from attributes.

Method/Property/ Event	Details	In object	Description
editor	174	"Instances Object"	A reference to the eWebEditPro+XML ActiveX control.
editorGetMethod	182	"Parameters Object"	Lets you save either the body only or the entire HTML document from the editor.
editorName	181	"eWebEditProUtility Object"	Holds the name of the editor that opened the popup.
ElementAttributes	72	"XML Data Object"	Returns valid attributes to the element delimited with the value specified in the strDelim string value.
ElementAttributeValues	73	"XML Data Object"	Retrieves the valid values defined in the schema for the given attribute in the current element.
ElementChildren	74	"XML Data Object"	Retrieves the valid elements defined in the schema for the current element.
ElementIsEmpty	75	"XML Data Object"	Returns true if the selected element is defined to be empty.
ElementMaxCount	75	"XML Data Object"	Returns the maximum times an element can appear within its parent element.
ElementMinCount	76	"XML Data Object"	Returns the minimum times an element can appear within its parent element
elemName	174	"Instances Object"	The name of the field element that contains the editor content.
embedAttributes	168	"Parameters Object"	Optional attributes to the EMBED tag.
EnableCreation	76	"Image Editor Object"	Enables or disables user interface that allows user to create new image.
EnableFormatChange	76	"Image Editor Object"	Enables or disables user's ability to change the file format and select the number of colors for image.
EnableNameChange	77	"Image Editor Object"	Enables or disables user's ability to change the name of image file.
End	165	"Button Tag Object"	Determines the end of the HTML that appears on the popup edit button.
ErrorClear	78	"Image Editor Object"	Clears any current errors.
ErrorCode	137	"XML Object"	Contains the error code for the last error.
ErrorDescription	78	"Image Editor Object"	Retrieves a text description of the last error encountered.
ErrorFilePos	137	"XML Object"	If error, position in the file where error occurred.
ErrorLine	137	"XML Object"	If error, line where error occurred.
ErrorPos	137	"XML Object"	If error, position in line where error occurred.
ErrorReason	138	"XML Object"	If error, contains a short description why there was an error.

Method/Property/ Event	Details	In object	Description
ErrorSrcText	138	"XML Object"	If error, the source within the file that causes the error.
ErrorURL	138	"XML Object"	If error, the URL of the loaded schema that contains the error.
ErrorValue	79	"Image Editor Object"	Returns a numeric value representing the last error encountered.
EstimateContentSize	79	"eWebEditPro Object"	Estimates the size of current content.
eWebEditProDblClickEl ement	193	"ewebeditproeve nts Object"	The JavaScript event that occurs when a user double-clicks a hyperlink, applet, object, image or table within the editor, unless a specific event handler for hyperlink, image or table is defined.
eWebEditProDblClickHy perlink	193	"ewebeditproeve nts Object"	The JavaScript event that occurs when a user double-clicks a hyperlink.
eWebEditProDblClickIm age	194	"ewebeditproeve nts Object"	The JavaScript event that occurs when a user double-clicks an image.
eWebEditProDblClickTa ble	194	"ewebeditproeve nts Object"	The JavaScript event that occurs when a user double-clicks a table.
eWebEditProExecCom mand	191	"ewebeditproeve nts Object"	The JavaScript defined in this function is called after an internal command is executed, or when an external command should be executed.
eWebEditProMediaNotifi cation		"ewebeditproeve nts Object"	
eWebEditProMediaSele ction	192	"ewebeditproeve nts Object"	Lets you add a media file handler.
eWebEditProReady	191	"ewebeditproeve nts Object"	Indicates it is safe to send commands to or access the MediaFile Object.
ExecCommand	81	"eWebEditPro ActiveX Control Object"	Causes the editor to perform the specified operation.
ExecCommand	80	"Image Editor Object"	Directly executes a command name with parameters, without going through eWebEditPro+XML 's command mechanism.
FileExistsLocally	81	"Parameters Object"	Determines if file exists on local system.
FileSize	151	"Parameters Object"	The size of the image file in bytes.
FileName	151	"Parameters Object"	The title of the file.
FileType	151	"Parameters Object"	The type of file.
FindDataField	81	"XML Object"	Returns CXMLData object.

Method/Property/ Event	Details	In object	Description
FirstCommand	82	"Command Item Object"	Sets the current reference to the first command available.
Focus	82	"eWebEditPro ActiveX Control Object"	Programmatically sets focus to eWebEditPro+XML editor using JavaScript.
FormatDelimitedAttribut es	82	"XML Data Object"	When attributes are contained in a delimited string that uses a pipe () as a delimiter, this method reformats them as they would appear in an element.
formName	174	"Instances Object"	The name or index of the form that contains this instance of the editor.
FWLoginName	152	"Parameters Object"	User's login name for the firewall. Not currently used.
FWPassword	152	"Parameters Object"	User's password for the firewall. Not currently used.
FWPort	152	"Parameters Object"	The firewall port to use for any transfer.
FWProxyServer	152	"Parameters Object"	Firewall proxy server. Not currently used.
FWUse	152	"Parameters Object"	If true , a firewall mechanism is used. Not currently used.
FWUsePassV	152	"Parameters Object"	If true , PASV mode FTP is enabled.
GetActiveStyleSheetTitl es	83	"eWebEditPro ActiveX Control Object"	Returns a comma-delimited list of the titles of active styles.
getBodyHTML	83	"eWebEditPro ActiveX Control Object"	Saves content within the BODY tags as HTML.
getBodyText	84	"eWebEditPro ActiveX Control Object"	Returns content text without formatting.
GetContent	85	"eWebEditPro ActiveX Control Object"	Retrieves specified content type from current edit session.
getDocument	85	"eWebEditPro ActiveX Control Object"	Saves entire HTML document currently in editor.
Get EnablePathResolution	153	"Parameters Object"	Enables path resolution functionality.
GetFieldValue	85	"Automatic Upload Object"	Reads value from the given data item.

Method/Property/ Event	Details	In object	Description
GetFileDescription	87	"Automatic Upload Object"	Returns description of file in list of files added for upload.
GetFileStatus	87	"Automatic Upload Object"	Retrieves upload status of file in list of files added for upload.
getHeadHTML	88	"eWebEditPro ActiveX Control Object"	Returns <HEAD> through </HEAD> HTML of current document as a string, including the HEAD tags.
GetImageInformation	89	"Image Editor Object"	Retrieves specified information about an image.
Get IsValid	153	"Parameters Object"	Returns whether current upload connection is valid.
getOpenerInstance	90	"eWebEditProUtility Object"	Returns a reference to Instance JavaScript object responsible for opening the popup.
getProperty	90	"Command Item Object"	Retrieves the property name given.
getProperty	90	"eWebEditPro ActiveX Control Object"	Reads from ActiveX control property.
getProperty	90	"Parameters Object"	Retrieves the property name given.
getPropertyBoolean	91	"Command Item Object"	Retrieves the property name given as a string.
getPropertyBoolean	90	"eWebEditPro ActiveX Control Object"	Returns value of a Boolean property.
getPropertyBoolean	91	"Parameters Object"	Retrieves the property name given as a string.
getPropertyInteger	91	"Command Item Object"	Retrieves the property name given as an integer.
getPropertyInteger	91	"eWebEditPro ActiveX Control Object"	Returns value of a Numeric property.
getPropertyInteger	91	"Parameters Object"	Retrieves the property name given as an integer.
getPropertyString	91	"Command Item Object"	Retrieves the property name given as a boolean.
getPropertyString	91	"eWebEditPro ActiveX Control Object"	Returns value of a String property.
getPropertyString	91	"Parameters Object"	Retrieves the property name given as a boolean.

Method/Property/ Event	Details	In object	Description
getSelectedHTML	92	"eWebEditPro ActiveX Control Object"	Returns currently selected content including any HTML tags.
getSelectedText	92	"eWebEditPro ActiveX Control Object"	Returns currently selected text with no formatting.
Get ShowResolutionOverride	153	"Parameters Object"	If true , user can manually enable or disable path resolution mechanism.
GetTagAttribute	92	"XML Data Object"	Retrieves the value of the given attribute within the currently selected tag.
GetValidFormats	92	"Image Editor Object"	Retrieves current set of valid file formats supported by feature.
Get WDDX	162	"eWebEditPro ActiveX Control Object"	Sets or retrieves assigned WDDX data.
GetXferType	153	"Parameters Object"	Retrieves or sets the transfer type string.
GetXPath	93	"Image Tag Object"	Returns XPath of selected Data Design field.
HandledInternally	153	"Parameters Object"	Determines if the upload has already been handled internally.
height	165	"Image Tag Object"	Determines the height of a popup window.
height	175	"Instances Object"	The height of the editor assigned when created.
HideAbout	93	"Toolbars Object"	Hides the about command button.
hideAboutButton	162	"eWebEditPro ActiveX Control Object"	Can remove the About button from the toolbar.
HideAllMenus	93	"Toolbars Object"	Hides all toolbar menus.
HorizontalSpacing	154	"Parameters Object"	Horizontal spacing attribute to use in HTML.
html	175	"Instances Object"	A string containing the HTML.
HTMLEncode	94	"eWebEditProUtility Object"	HTML encodes the given string.
Icon	144	"XML Data Object"	Contains the path of the icon shown with the tag description.

Method/Property/ Event	Details	In object	Description
id	175	"Instances Object"	The name of the eWebEditPro+XML editor element in the object (Internet Explorer) or embed (Netscape) tag.
ImageEditor	94	"Image Editor Object"	Retrieves Image Edit object that exists within WebImageFX.
ImageError	184	"Image Editor Object"	Notifies client application or script that error has occurred.
ImageHeight	154	"Parameters Object"	The height of the image.
ImageWidth	154	"Parameters Object"	The width of the image.
insertMediaFile	94	"Instances Object"	Inserts an image file (or other media file) to the editor.
installPopup	178	"eWebEditPro Object"	If true , a window with the intro.htm page pops up.
instances collection	178	"eWebEditPro Object"	An array of in-line editor objects of type eWebEditProEditor or eWebEditProAlt.
IsAttributeRequired	95	"XML Data Object"	Returns true if the given attribute is defined as required in the selected element.
isAutoInstallSupported	178	"eWebEditPro Object"	If true , eWebEditPro+XML can be automatically installed.
isChanged	95	"eWebEditPro Object"	Determines if editor content has changed.
isChanged	95	"Instances Object"	Returns true if content in any editor on the page was modified.
IsDataField	96	"XML Data Object"	Returns true if current tag is a Data Design tag.
IsDirty	162	"eWebEditPro ActiveX Control Object"	Returns true if content has changed.
IsDirty	96	"Image Editor Object"	Returns a non-zero (boolean true) value if user modified image.
isEditor	96	"eWebEditPro Object"	Indicates if an instance of an editor exists by the given name, and if the instance has a valid 'editor' property.
isEditor	97	"Instances Object"	Returns true if the .editor object is available.
isEditorReady	97	"eWebEditPro ActiveX Control Object"	If "true", editor is ready to process a command.
isInstalled	179	"eWebEditPro Object"	If "true", eWebEditPro+XML is installed.

Method/Property/ Event	Details	In object	Description
IsLocal	154	"Parameters Object"	Set to true if a local file will be placed into the SrcFileName property.
isOpenerAvailable	99	"eWebEditProUtility Object"	Determines if page that opened the popup is still open.
IsPresent	99	"Image Editor Object"	Returns true if WebImageFX is installed properly on client system.
isSupported	179	"eWebEditPro Object"	If true , eWebEditPro+XML is supported in this environment. It may not be installed yet.
IsTagApplied	99	"eWebEditPro ActiveX Control Object"	Indicates if a specified XML tag can be applied at the current cursor location.
IsValid	100	"Command Item Object"	Returns "true" if the interface references a valid command.
IsValid	100	"XML Data Object"	Offers a quick check to determine if the tag retrieved with ActiveTag is a valid tag.
isVisible	101	"Image Editor Object"	This method returns true if WebImageFX is visible to user from within eWebEditPro+XML .
languageCode	181	"eWebEditProUtility Object"	The language code of the browser.
License	162	"eWebEditPro ActiveX Control Object"	The license keys of the editor.
ListCommandName	101	"Command Item Object"	Returns the name of the command associated with the item at the index specified.
ListFilesWithStatus	101	"Automatic Upload Object"	Retrieves a list of files with a specified status.
ListLength	105	"XML Data Object"	Returns the number of items in the list.
load	105	"eWebEditPro Object"	Loads content into all in-line editors on page from standard HTML elements with the same name.
load	104	"Instances Object"	Loads content into editor.
LoadedFileName	105	"Image Editor Object"	Returns name of loaded image file.
LoadingImage	185	"Image Editor Object"	Notifies client application or script that image file has loaded.
LoadList	105	"XML Data Object"	Creates the list of items from an XML document.
locale	106	"Parameters Object"	Specifies the locale file to use.

Method/Property/ Event	Details	In object	Description
Locale	162	"eWebEditPro ActiveX Control Object"	The URL of the localization directory or file.
LoginName	146	"Automatic Upload Object"	The login name of the user uploading the image.
LoginName	154	"Parameters Object"	The login name of the user uploading the image.
LoginRequired	147	"Automatic Upload Object"	Enables or disables the act of logging into a remote site.
maxContentSize	175	"Instances Object"	The largest number of characters that can be saved in the editor window.
maxContentSize	168	"Parameters Object"	The largest number of characters that can be saved in editor.
MaxFileSizeK	155	"Parameters Object"	Maximum size in kilobytes of image to be uploaded.
MaxListboxWidth	136	"Command Item Object"	Sets or retrieves the width of an edit box or a list box in characters.
MediaFile	106	"eWebEditPro ActiveX Control Object"	Returns reference to the Media File object.
MediaType	155	"Media File Object"	Determines which valid extensions are provided in the Media File Selection dialog.
MinChildElementCount()	107	"XML Data Object"	Returns the minimum number of times a child element can exist under the current element.
name	176	"Instances Object"	The name assigned to this instance of the editor when it was created.
NeedConnection	155	"Parameters Object"	Determines if a connection is necessary with the current upload method.
NextCommand	108	"Command Item Object"	Sets the current reference to the next command available.
objectAttributes	169	"Parameters Object"	Optional attributes to the OBJECT tag.
onbeforeedit	188	"Event Object"	Occurs when the user clicks the button created by the createButton method.
onbeforeedit	188	"eWebEditPro Object"	Occurs when the onbeforeedit method is invoked.
onbeforeload	189	"Event Object"	Occurs when the load method is invoked.
onbeforeload	189	"eWebEditPro Object"	Occurs when the load method is invoked.

Method/Property/ Event	Details	In object	Description
onbeforeload	189	"Instances Object"	Occurs when the load method is invoked.
onbeforesave	189	"Event Object"	Occurs when the save method is invoked.
onbeforesave	189	"eWebEditPro Object"	Occurs when the save method is invoked.
onbeforesave	189	"Instances Object"	Occurs when the save method is invoked.
onblur	187	"eWebEditPro ActiveX Control Object"	An event that fires when the editor loses the focus.
onblur	187	"Parameters Object"	An event that fires when the editor loses the focus.
oncreate	187	"Event Object"	Occurs when the create method is invoked.
oncreate	187	"eWebEditPro Object"	Occurs when the create method is invoked.
oncreatebutton	188	"Event Object"	Occurs when the createButton method is invoked.
oncreatebutton	188	"eWebEditPro Object"	Occurs when the createButton method is invoked.
ondblclickelement	186	"eWebEditPro ActiveX Control Object"	Double-clicking on a hyperlink, applet, object, image, or table causes this event to fire.
ondblclickelement	186	"Parameters Object"	The JavaScript event that occurs when a user double-clicks any selectable element object.
onedit	188	"Event Object"	Occurs after the popup window closes.
onedit	188	"eWebEditPro Object"	Occurs after the popup window closes.
onerror	190	"Event Object"	Occurs when an error occurs because the save method failed.
onerror	190	"eWebEditPro Object"	Occurs when an error occurs because the save method failed.
onerror	190	"Instances Object"	Occurs when an error occurs because the save method failed. <i>See Also:</i> " The onerror Event "
onexeccommand	186	"eWebEditPro ActiveX Control Object"	Raised after a toolbar button is pressed, a toolbar dropdown list item is selected, or a context menu (right-click menu) item is selected.
onexeccommand	187	"Parameters Object"	The default JavaScript onexeccommand handler.
onfocus	187	"Parameters Object"	An event that fires when the editor gains the focus.

Method/Property/ Event	Details	In object	Description
onfocus()	187	"eWebEditPro ActiveX Control Object"	An event that fires when the editor gains the focus.
onload	190	"Event Object"	Occurs when the load method is complete.
onload	190	"eWebEditPro Object"	Occurs when the load method is complete.
onload	190	"Instances Object"	Occurs when the load method is complete.
onready	190	"Event Object"	Occurs when it is safe to send commands or access the Media File Object.
onready	190	"eWebEditPro Object"	Occurs when it is safe to send commands or access the Media File Object.
onsave	189	"Event Object"	Occurs when the save method is complete.
onsave	189	"eWebEditPro Object"	Occurs when the save method is complete.
onsave	189	"Instances Object"	Occurs when the save method is complete.
ontoolbarreset	189	"Event Object"	Occurs when the toolbar is initialized or reset.
ontoolbarreset	189	"eWebEditPro Object"	Occurs when the editor's toolbar is initialized or reset.
openDialog	108	"eWebEditPro Object"	Opens the popup Web page specified by fileName.
outerXML	109	"XML Data Object"	Returns the XML of the custom tag as a string, for example <mytag>some text</mytag>.
parametersobject	179	"eWebEditPro Object"	An object of type eWebEditProParameters containing the default set of parameters used when creating an instance of the editor or button.
Password	147	"Automatic Upload Object"	The password of the user uploading the image.
Password	155	"Parameters Object"	The password of the user uploading the image.
pasteHTML	109	"eWebEditPro ActiveX Control Object"	Replaces selected content with string passed to pasteHTML.
pasteText	110	"eWebEditPro ActiveX Control Object"	Replaces selected content with string passed to pasteText.
path	169	"Parameters Object"	The path to the eWebEditPro+XML files relative to the hostname.

Method/Property/ Event	Details	In object	Description
PopulateTagsWithStyles	110	"eWebEditPro ActiveX Control Object"	Applies current, active styles to content's tags.
popup	171	"Parameters Object"	Lets you pass four parameters to the popup Web page.
PopupMenu	111	"Toolbars Object"	Brings up a popup menu.
Port	149	"Automatic Upload Object"	The port used for HTTP posting or FTP transfer.
Port	155	"Parameters Object"	The port to use for uploads.
preferredType	169	"Parameters Object"	Specifies the type of editor to create.
ProxyServer	156	"Parameters Object"	The name of the proxy server to use with uploads.
PublishHTML	111	"Image Editor Object"	Formats named values into HTML tag that contains attribute/value combinations.
query	173	"InstallPopup Object"	An optional parameter that specifies query string values to pass to the page specified by URL parameter.
query	173	"Popup Object"	A query to pass parameters to the popup window.
queryArgs	181	"eWebEditProUtility Object"	The array of URL query string parameters passed to the page.
ReadNamedData	112	"Automatic Upload Object"	Retrieves the data value of the data name from the file specified.
readOnly	176	"Instances Object"	Prevents user from modifying editor content.
readOnly	170	"Parameters Object"	Prevents the user from modifying the editor content.
ReadOnly	162	"eWebEditPro ActiveX Control Object"	Prevents user from modifying editor content.
ReadResponseHeader	112	"Automatic Upload Object"	Retrieves the header of the response sent by the server.
ReadUploadResponse	113	"Automatic Upload Object"	Reads the full text returned from the server as a response to the upload.
receivedEvent	176	"Instances Object"	"True" if an event has been received from ActiveX control.
refreshStatus	113	"eWebEditPro Object"	Updates the value of several properties such as status, isIE, and isNetscape,

Method/Property/ Event	Details	In object	Description
relocate	113	"Parameters Object"	Relocates the 'on' event handlers to point to the frame where the functions are defined.
RemotePathFileName	156	"Parameters Object"	The remote path and name of the currently selected file.
RemoveFieldValue	114	"Automatic Upload Object"	Removes given data item so it is not sent with the upload.
RemoveFileForUpload	114	"Automatic Upload Object"	Removes a specified file from the list of files for uploading.
RemoveListItem	115	"XML Data Object"	Removes an item from the list.
RemoveNamedData	115	"Automatic Upload Object"	Removes the named data set from the file specified.
reset	116	"Parameters Object"	Reinitializes all values to the default defined in eWebEditProDefaults (ewebeditprodefaults.js).
ResolveMethod	156	"Parameters Object"	The method by which the image source path is resolved.
resolvePath	116	"eWebEditPro Object"	Prepends the URL with the eWebEditPro path.
ResolvePath	156	"Parameters Object"	The path used to resolve an image path when GIVEN is the resolution method.
RetrieveHTMLString	116	"Parameters Object"	Returns HTML string to be used for insertion into HTML.
RootTag	139	"XML Object"	Retrieves or sets the root tag of a loaded XML document.
rows	170	"Parameters Object"	The number of rows in the TEXTAREA element if eWebEditPro+XML is not installed or not supported.
save	117	"eWebEditPro Object"	Saves content into all in-line editors on page from standard HTML elements with the same name.
save	116	"Instances Object"	Saves content.
Save	116	"Image Editor Object"	Saves currently edited image with currently selected file parameters.
SaveAs	117	"Image Editor Object"	Saves the currently edited image with the specified parameters.
SavedFileName	117	"Image Editor Object"	Returns name that file was actually saved as.
SavingImage	185	"Image Editor Object"	Called before current image is saved to local file system.
SchemaExists	118	"XML Object"	Allows client to check if the schema assigned the given namespace is loaded.

Method/Property/ Event	Details	In object	Description
SeparatorBarAdd	118	"Toolbars Object"	Adds a separator bar to the specified toolbar.
SeparatorSpaceAdd	118	"Toolbars Object"	Adds a separator space to the specified toolbar.
ServerName	146	"Automatic Upload Object"	Specifies the server to use with the receiving page.
SetConfig	119	"Image Editor Object"	Specifies which configuration file to use for controlling WebImageFX.
SetContent	119	"eWebEditPro ActiveX Control Object"	Assigns given content to the editor session.
setDocument	120	"eWebEditPro ActiveX Control Object"	Replaces entire document with specified document.
SetFieldValue	121	"Automatic Upload Object"	Assigns a data item to be sent with the file.
SetFileDescription	121	"Automatic Upload Object"	Sets description of specified file.
SetFileStatus	122	"Automatic Upload Object"	Sets status of given file.
setHeadHTML	119	"eWebEditPro ActiveX Control Object"	Sets <HEAD> through </HEAD> portion of the document header.
SetLocale	123	"Image Editor Object"	Specifies a Locale translation file to use.
setProperty	124	"Command Item Object"	Sets the named property to the value given.
setProperty	124	"eWebEditPro ActiveX Control Object"	Writes to ActiveX control property.
setProperty	124	"Parameters Object"	Sets the named property to the value given.
SetTagAttribute	125	"XML Data Object"	Sets the given attribute value to the value sent to the method.
SetValidFormats	125	"Image Editor Object"	Specifies a set of formats that are considered valid by a client application or script.
ShowAbout	126	"Toolbars Object"	Shows the about button
ShowActiveStylesDetails	126	"eWebEditPro ActiveX Control Object"	Returns a comma-delimited list of the active style sheet titles and style information

Method/Property/ Event	Details	In object	Description
ShowAllMenus	127	"Toolbars Object"	Restores the view of menus hidden with HideAllMenus.
ShowHeight	157	"Parameters Object"	The height attribute of the HTML image tag.
ShowIcon	144	"XML Data Object"	Contains whether an icon is shown to the user with the tag.
ShowName	144	"XML Data Object"	Determines if the name, or element description, is shown with the element.
ShowRootTag	139	"XML Object"	Determines if the user can see root tag in the header of the XML data.
ShowWidth	157	"Parameters Object"	The width attribute for the HTML image tag.
src	165	"Image Tag Object"	Determines the source of the image that appears on the button used to open the popup window.
SrcFileName	157	"Parameters Object"	The full location of the source file.
srcName	167	"Event Object"	The name of the instance of the editor that is the source of the current event.
SrcPath	163	"eWebEditPro ActiveX Control Object"	Specifies where eWebEditPro+XML is installed.
Start	165	"Button Tag Object"	Determines the beginning of the HTML that appears on the popup edit button.
status	179	"eWebEditPro Object"	Reflects the current state of eWebEditPro+XML .
status	176	"Instances Object"	The status of this editor.
StyleSheet	163	"eWebEditPro ActiveX Control Object"	Specifies style sheet file (CSS) to apply to editor content.
tagAttributes	166	"Button Tag Object"	Used to assign custom attributes to the popup edit button.
TagCount	127	"eWebEditPro ActiveX Control Object"	Indicates how many times a specified XML tag exists in the content.
TagName	145	"XML Data Object"	Contains the name of the element.
TagStyle	145	"XML Data Object"	Contains the style parameters used to display the tag.
TagType	145	"XML Data Object"	Contains the type of flag that defines how the element is arranged when formatted.

Method/Property/ Event	Details	In object	Description
textareaAttributes	170	"Parameters Object"	Optional attributes to the TEXTAREA tag.
Thumbnail	127	"Image Editor Object"	Creates a thumbnail of the current image or a specified image file.
Title	163	"eWebEditPro ActiveX Control Object"	A document title for page.
ToolbarAdd	129	"Toolbars Object"	Creates a toolbar and adds it to the toolbars available to the user.
ToolbarModify	129	"Toolbars Object"	Modifies an existing toolbar.
Toolbars	130	"eWebEditPro ActiveX Control Object"	Returns a reference to the Toolbar Interface object.
TransferMethod	157	"Automatic Upload Object"	Specifies how the Automatic Upload mechanism performs an upload when local files are detected.
TransferMethod	157	"Parameters Object"	The name of the upload method used if the ProvideMediaFile method is called.
TransferRoot	147	"Automatic Upload Object"	The destination path where the image will be placed.
TransferRoot	158	"Parameters Object"	The destination path where the image will be placed.
Transform	130	"XML Object"	Performs a transformation on the document content given to the method.
TransformOnLoad	140	"XML Object"	Sets or reads the XSLT file used when whole content is loaded.
TransformOnSave	140	"XML Object"	Sets or reads XSLT file used when whole content is extracted from the editor.
type	166	"Button Tag Object"	Determines the form of the popup edit button.
type	167	"Event Object"	The name of the current event without the "on" prefix.
type	176	"Instances Object"	Indicates which type of editor was created on page.
upgradeNeeded	180	"eWebEditPro ActiveX Control Object"	If true, an older version eWebEditPro+XML is installed and needs to be upgraded.
UploadConfirmMsg	131	"Automatic Upload Object"	Sets user message displayed on the user intervention dialog.
url	172	"InstallPopup Object"	Specifies URL of Web page to display in popup window when an automatic installation is expected.

Method/Property/ Event	Details	In object	Description
url	173	"Popup Object"	The URL to the Web page that contains the editor that appears in the popup window.
UseHTMLString	131	"Parameters Object"	Information from given HTML string is placed into the appropriate Media object properties.
UsePassV	158	"Parameters Object"	If true, FTP works in passive mode.
Validate	132	"XML Object"	Validates a given set of content.
ValidChildElement	133	"XML Data Object"	Offers a quick check to determine if a given element is valid within the current element.
ValidConnection	158	"Parameters Object"	If true , system made valid connection with current connection parameters.
ValidExtensions	147	"Automatic Upload Object"	The file extensions of images that can be uploaded, entered as a comma-delimited string.
ValidExtensions	158	"Parameters Object"	File extensions of images that can be uploaded.
value	166	"Button Tag Object"	Determines the value of the popup edit button.
Version	180	"eWebEditPro Object"	The version of the control.
versionInstalled	163	"eWebEditPro ActiveX Control Object"	Retrieves the version of the control.
VerticalSpacing	158	"Parameters Object"	The value of the vertical spacing attribute of the HTML image tag.
WebPathName	159	"Parameters Object"	The Web accessible name of the specified file.
WebRoot	147	"Automatic Upload Object"	The base location for accessing uploaded images from a Web page.
width	165	"Image Tag Object"	Determines the width of a popup window.
width	177	"Instances Object"	The width of editor assigned when created.
windowFeatures	172	"InstallPopup Object"	Specifies popup window features as defined for standard JavaScript window.open() method.
windowFeatures	173	"Popup Object"	The parameters passed to the standard JavaScript window.open() method.
windowName	172	"InstallPopup Object"	Specifies the name of the popup window.
windowName	174	"Popup Object"	The name assigned to the popup window created by the standard JavaScript function window.open().

Method/Property/ Event	Details	In object	Description
XmlHeader	141	"XML Object"	An alternative, and more direct, method of accessing XML header information.
xmllInfo	164	"eWebEditPro ActiveX Control Object"	Dynamically assigns XML and custom tag configuration data that is external to normal configuration data.
XMLProcessor	133	"eWebEditPro ActiveX Control Object"	Retrieves interface to XML Object (only available with eWebEditPro+XML).

Master List of Methods

Method: ActiveTag

Description: Retrieves a reference to a CXMLData object that contains the tag that is applied to the current selection.

See Also: "Method: IsValid"

Object: "XML Object"

Type: CXMLData Object

Parameters: none

Example: This example shows how to retrieve the name of the currently active element.

```
function retrieveCurrentTagName(sEditorName)
{
    var objXML = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXML.ActiveTag();
    return(objXML.getPropertyString("TagName"));
}
```

Method: addEventHandler

Description: Defines event handlers for **eWebEditPro+XML** events, such as onready.

Instead of setting `eWebEditPro.onready = your_onready_handler`, which replaces any handler that may have been assigned, use the following:

```
eWebEditPro.addEventHandler("onready", your_onready_handler);
```

This method adds an event handler to a list of handlers that are called when the onready event fires. The generic syntax is:

```
object.addEventHandler(event_name, event_handler)
```

Parameters: the name of the event, the event handler

Object: "eWebEditPro Object" and "Instances Object"

Method: AddFileForUpload

Description: Adds a file to the list of files to upload. This file does not need to exist and does not need to be in the content. When a file is added, the status is set to 1.

See Also: ["Method: ListFilesWithStatus"](#)

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
LocalFileName	String	The name and path of the local file to upload.
Description	String	The description of the file.

Example

```
objAuto.AddFileForUpload(strLocalFile, strDescription)
objAuto.AddFileForUpload("C:\My Pictures\images\me.gif, "A picture of me last weekend.")
```

Return: None

Method: addInlineStyle

Description: Adds an inline <STYLE> . . . </STYLE> to the document header.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
strReturnValue = eWebEditPro.Editor1.addInlineStyle (strSelector, strStyle)
```

Parameters

strReturnValue - If successful, strReturnValue is equal to strStyle. If unsuccessful, strReturnValue is an error message.

strSelector - The tag to which the strStyle is applied. Note that the strSelector should not represent more than one tag. To apply the same style to multiple tags, add a style for each tag.

strStyle - The CSS syntax style to apply to the strSelector tags in the content.

Remarks

The new style sheet overrides rules for existing tags. For example, if a style sheet affects P, LI and DIV, and there is a call to addInlineStyle that "adds" a style for the P tag, the new P style overrides the existing P style, but the LI and DIV styles remain in effect.

The strStyle syntax starts and ends with the style information. The function supplies the curly brackets that surround the style information. For example:

```
strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")
```

Example

This adds style H4, identified by the style title "UserH4," to the document header.

```
strNewStyle = eWebEditPro.Editor1.addInlineStyle("H4", "font-size:22pt; margin:15px; color:blue; font-family:Century Gothic", "UserH4")
```

As a result, the header HTML now has this extra content.

```
<STYLE title=UserH4>H4 {COLOR: blue; FONT-FAMILY: "Century Gothic"; FONT-SIZE: 22pt; MARGIN: 15px}
```

Method: AddItem

Description: In an edit control, this method sets the text. In a list box, it adds an item to the dropdown list. Otherwise, it does nothing.

Object: "Command Item Object"

Parameters

Parameter	Type	Description
ItemText	String	The text of the selection.
ItemData	Long	Data associated with the command. If this is omitted or 0 (zero), the data returned with the selection is the 0-based index into the list.
StrCmdName	String	Command to associate with the list selection. If this is a value, the specified command name is sent to the client in place of the command that contains the list.

Return: Nothing

Method: addLinkedStyleSheet

Description: Adds a linked style sheet reference to the document header.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
strReturnValue = eWebEditPro.Editor1.addLinkedStyleSheet(strURL)
```

Parameters

Parameter	Description
strReturnValue	If successful, strReturnValue is equal to strURL. If unsuccessful, strReturnValue is blank.
strURL	The URL of the style sheet to link to.

Remarks

The new style sheet overrides rules for existing tags. For example, if a style sheet affects P, LI and DIV, and there is a call to addLinkedStyle that “adds” a style for the P tag, the new P style overrides the existing P style, but the LI and DIV styles remain in effect.

Example

```
strMyStyleReturn = eWebEditPro.Editor1.addLinkedStyleSheet("http://www.ourcompany.com/styles/mystyles.css")
```

As a result, the header HTML now has this extra content.

```
<LINK href="http://www.ourcompany.com/styles/mystyles.css" rel=stylesheet title= http://www.ourcompany.com/styles/mystyles.css>
```

Method: AddListItem

Description: Adds an item to the end of the list. See *Also:* ["Specifying Items in a Select List Field"](#)

Object: "XML Data Object"

Parameters:

Parameter	Description
strCaption	the display text
strValue	the value if selected

Return: error code. 0=no error.

Example

JavaScript example

```
objDDField.AddItem("Agree", "1");
objDDField.AddItem("Disagree", "-1");
objDDField.AddItem("Don't know", "0");
```

Method: AddNamedData

Description: Adds the named data set to individual upload files in the file store.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
filename	string	The filename in the file store to which the named data set is added.
data name	string	The name/id of the named data set.
data value	string	the value/data of the named data set.

Example

```
objAuto.AddNamedData(sFileName, sDName, sDValue);
```

or

```
objAuto.AddNamedData("c:\abc.jpg", "id", "123");
```

Return: boolean

Method: AddSchema

Description: This method loads a schema file. You can add any number of schemas.

The last schema loaded becomes the active schema, so adding a schema also selects it.

If the namespace is empty, the location of the schema is used. If a namespace is specified in the schema, that namespace is used in place of the requested namespace.

Since the namespace can be different than requested, after a schema is loaded, the namespace should be retrieved using the ActiveSchema property to ensure that you have the correct namespace for the loaded schema.

See Also: "Working with Schemas" on page 666

Object: "XML Object"

Return: Boolean - If true, the load was successful. Check the error properties for errors.

Parameters:

Name	Type	Description
strSchemaLoc	String	<p>The path to the schema file. This can be either a relative or a fully qualified path.</p> <p>The path must point to a file -- it cannot be a stream like what is allowed with XML data.</p> <p>The file must be accessible to the eWebEditPro+XML session.</p>
strNameSpace	String	<p>The namespace to associate with this schema. If blank, the location of the schema is used. If the namespace is specified in the schema, that namespace is used instead.</p> <p>Use the ActiveSchema property to retrieve the final namespace attached to this schema.</p>

Example:

```
function GetXMLProcessor(sEditorName)
{
    return(eWebEditPro.instances[sEditorName].editor.XMLProcessor());
}
function LoadSchema(sEditorName, sSchemaLocation, sNameSpace)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    objXmlDoc.AddSchema(sSchemaPath, sNSTarget);
    if(0 != objXmlDoc.getPropertyInteger("ErrorCode"))
    {
        alert("Error: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

Method: AnySchemasLoaded

Description: This method determines if any schemas are currently loaded.

See Also: ["Working with Schemas" on page 666](#)

Object: ["XML Object"](#)

Parameters: None

Return: Boolean - True if any schemas are loaded.

Example:

```

function OfferElementAttributes(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();

    if(objXmlDoc.AnySchemasLoaded())
    {
        ShowAnyValidAttributes(sEditorName);
    }
    else
    {
        alert("There are no schemas loaded. Please load a schema to provide valid attributes.");
    }
}

```

Method: ApplyTag

Description:

WARNING! No changes made to an XML Data tag are reflected in the content until this tag is applied.

Applies the given CXMLData object information to the currently accessed inner tag in the content of the object. All information within the XML Data object is used to create the tag and replace the currently active tag. This includes any tag content contained within the object.

So, you can use this method to change an entire document. If the root tag is the current tag and the tag is replaced, the entire document is changed to match the contents of the tag object.

When a change is made to an XML Data object that contains a selected tag, those changes are not immediately seen in the content. Instead, the changes are stored in the object until the object is applied. This transfers all changes into the content. The changes need to be stored and then applied at once for performance reasons.

Object: "XML Object"

Return: Boolean - **True** if the tag object could be applied.

Parameters:

Name	Type	Description
objTagData	XML Data	The XML Data object that contains the tag information. The currently active tag is completely replaced with the given tag.

Example: You must apply a tag to the document before any changes made to the tag are reflected in the document's content. This is very important.

```
objXmlDoc.ApplyTag(objTag);
```

Method: AskOpenFile

Description: Displays a dialog that prompts the user to select an image to edit.

Object: "Image Editor Object"

Parameters

None

Remarks

This method lets the client application or script externally bring up the Open File dialog.

Return: Boolean - The status of bringing up the dialog.

A **True** value means the dialog was successful. Otherwise, there was an error. A cancel does not count as an error.

Method: AskSaveAs

Description: Displays a dialog that asks the user to select a format and file name for the current image.

Object: "Image Editor Object"

Parameters: None

Remarks

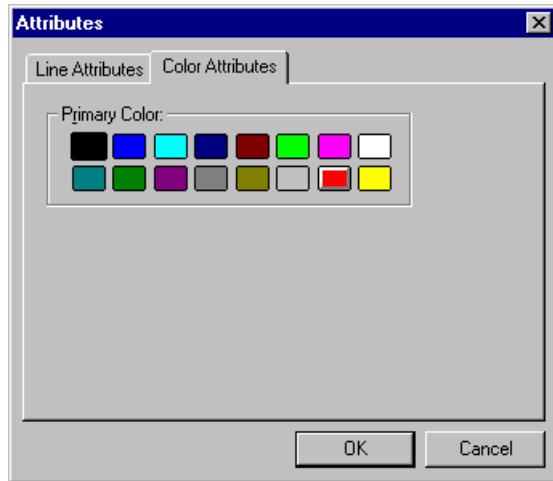
This method offers the client application or script the ability to display the "Save As" dialog to the user.

Return: String - the full file name that the user saved as the image. An empty string denotes an error or a cancel.

Method: AskSelectColor

Description: Displays a dialog in which the user can choose a color and line size of a recently-drawn annotation.

Object: "Image Editor Object"



Parameters

None

Method: AttributeValueDefault

Description: Returns the default value defined in the schema for the given attribute in the current element. If there is no default and a list of valid values is defined in the schema, the first item in the list is returned. Otherwise, the return value is empty.

Object: "XML Data Object"

Return: String - The default value for the attribute.

Parameters:

strAttribute	String	The attribute whose default value you are retrieving.
--------------	--------	---

Example:

This example processes all of an attribute's valid values and shows which one is the default.

```

function ShowValidAttributeValuesWithDefault(sEditorName, sAttribute)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idData = 0
    var strVals = "";
    var strDefVal = "";

    strVals = objElem.ElementAttributeValues(sAttribute, "|");
    if(0 != strVals.length)
    {
        var aryData = strVals.split("|");
        strDefVal = objTag.AttributeValueDefault(sAttribute);
        for(idData = 0; idData < aryData.length; idData++)
        {
            if(strDefVal == aryData[idData])
            {
                alert("Attribute Value: " + aryData[idData] + " [ default ]");
            }
            else
            {
                alert("Attribute Value: " + aryData[idData]);
            }
        }
    }
    else
    {
        alert("There are no defined values for " + sAttribute);
    }
}

```

Method: autoInstallExpected

Description: Returns a boolean that indicates whether an automatic download and installation of **eWebEditPro+XML** is expected or not. This value can be used to display a message informing the user what to expect while **eWebEditPro+XML** is installed.

See Also: “Client Installation Pages” on page 297

Parameters

Parameter	Description
true	Automatic installation is supported and eWebEditPro+XML is either not installed or requires upgrading.
false	Either automatic installation is not supported or the correct version of eWebEditPro+XML is installed.

Method: AvailableSchemas

Description: This method retrieves all of the namespaces of the loaded schemas in a delimited string.

See Also: ["Working with Schemas" on page 666](#)

Object: "XML Object"

Return: String - The list of namespaces for loaded schemas.

Parameters:

Name	Type	Description
strDelim	string	The delimiter to use for the string of namespaces. If this is an empty string, the pipe character () is used.

Example: This example lists all of the loaded schemas.

```
function ListLoadedSchemas(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var strVals = objXmlDoc.AvailableSchemas(" | ");
    var strActive = objXmlDoc.getPropertyString("ActiveSchema");

    // The method objXmlDoc.AnySchemasLoaded() could also have been
    // used here to determine if any shemas are loaded, but checking
    // the length of the returned string will do the same thing, and
    // we have gone to the trouble of retrieving it, so we will use
    // it to check for any valid schemas.
    if(strVals.length > 0)
    {
        var aryVals = strVals.split(" | ");
        var idx = 0;

        for(idx = 0; idx < aryVals.length; idx++)
        {
            if(aryVals[idx] == strActive)
            {
                alert("Loaded Schema: " + aryVals[idx] + " ** Active **");
            }
            else
            {
                alert("Loaded Schema: " + aryVals[idx]);
            }
        }
    }
    else
    {
        alert("There are no schemas loaded.");
    }
}
```

Method: BodyStyle

Description: Sets/gets the document's body style.

BodyStyle adds an inline style to the document header. It does *not* add attributes to the Body tag.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
eWebEditPro.Editor1.bodyStyle = strCssText
```

Parameters Set

New_BodyStyle - The CSS style (without curly braces) for the new body style.

Parameters Get

(return value) - The CSS of the current body style.

Example

The following creates an inline body style that sets the document font to red Arial.

```
eWebEditPro.Editor1.bodyStyle = "color:red;font-family:Arial"
```

strBodyStyle now looks like this.

```
BODY {  
    COLOR: red; FONT-FAMILY: Arial  
}
```

Method: BuildErrorDescEng

Description: Retrieves a full error description in English. This description contains the error code, description, file location, and any other information to aid in understanding the error.

The error could also be assembled by the client using the other error methods.

Object: "XML Object"

Return: String - A detailed description of the current error.

Parameters: None

Example:

```
alert(objXmlDoc.BuildErrorDescEnd());
```

Method: CanInsert

Description: This method returns a value that indicates whether or not a user can insert a tag at the current position.

Object: "XML Data Object"

Return: Boolean

True means that tags or contents can be inserted at the current cursor location of the retrieved tag.

False means that data cannot be inserted at the cursor location. This would happen if

- the cursor is in a location in the tag that cannot accept content, such as the description area, or
- the current tag is marked as uneditable

Parameters: None

Example:

```
function InsertElementData(sEditorName, sElementData)
{
    var objElem = GetCurrentXMLTag(sEditorName);
    if(true == objElem.CanInsert())
    {
        eWebEditPro.instances[sEditorName].editor.PasteHTML(sElementData);
        //alert("Insert is possible.");
    }
    else
    {
        alert("Can't insert at this location.");
    }
}
```

Method: Clear

Description: In a list box, this method clears all entries. In an edit box, it clears the text. In a toggle, it ensures that it is un-toggled.

Object: "Command Item Object"

Parameters: none

Return: Nothing

Method: ClearList

Description: Removes all the items from list. See Also: "[Specifying Items in a Select List Field](#)"

Object: "XML Data Object"

Parameters:

none

Return: error code. 0=no error.

Example:

Javascript example

```
objDDField.ClearList();
```

Method: ClearStylesFromTags

Description: Removes the style attribute from all tags in the document.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
eWebEditPro.Editor1.clearStylesFromTags
```

Parameters

none

Example

Given the style sheet added inline and the call to PopulateTagsWithStyles:

```
Dim bResult As Boolean  
strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css")  
bResult = eWebEditPro1.PopulateTagsWithStyles
```

The resulting HTML looks like this.

```
<H1 style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: 'Arial'; FONT-SIZE: 11pt; MARGIN: 0in">This  
text is styled by testpage3.css</H1>  
<H2 style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: 'Arial'; FONT-SIZE: 10pt; MARGIN: 0in">This  
text is styled by testpage3.css</H2>
```

Calling ClearStylesFromTags removes the styles and produces:

```
<H1>This text is styled by testpage3.css</H1>  
<H2>This text is styled by testpage3.css</H2>
```

Method: CmdFirst

Description: Sets the command object to look at the first command in the menu or toolbar. All methods in the CCommandItem interface apply to that command. You must use the object's other properties and methods to obtain information on the command.

This method works with CmdNext.

Object: "Command Item Object"

Method: CmdNext

Description: Sets the command object to look at the next command in the menu or toolbar. All methods in the CCommandItem interface will apply to that command. You must use the object's other properties and methods to obtain information on the command.

A return value of "false" means there is no next command. If "false", the command reference does not change and remains on the previous command.

This method works with CmdFirst.

Here is an example of how to use these methods.

```
function ListAllMenuItems(sEditorName)
{
    var objMenus = eWebEditPro.instances[sEditorName].editor.Toolbars();
    var objCmdItem = objMenus.CommandItem(""); // no command name for no specific command

    if(objCmdItem.CmdFirst() == true)
    {
        do
        {
            ShowText (objCmdItem.getPropertyString("CmdName") + " - " +
                      objCmdItem.getPropertyString("CmdCaption"));
        } while(objCmdItem.CmdNext() == true);
    }
}
```

Object: "Command Item Object"

Method: CommandAdd

Description: Adds a command to the specified toolbar.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
CommandName	String	The name of the command to add. When selected, this is the string value sent up as the command. <i>See Also:</i> " Commands "
CommandCaption	String	The caption to use next to the command.
ToolTip	String	Tool tip text that pops up when the cursor hovers over a command.
ImageFile	String	The file to use as the icon image. This can also be one of the internal image definitions.
Options	Long	Bit field of etbToolbarOptions bits describing specific options for the toolbar. <i>See Also:</i> " etbToolbarOptions ".
Style	Long	The style from the etbCommandStyles set of values. <i>See Also:</i> " etbCommandStyles ".

ToolbarName	String	The name of the toolbar to attach this command to. If left blank, it is not assigned to a toolbar but is available for customization.
-------------	--------	---

Return:

This command returns a reference to the command item that was created. Be sure to check that the command is *not* nothing (that is, null) before using it.

Method: CommandDelete

Description: Deletes a command from a toolbar. If a toolbar name is not specified, it is deleted from all locations.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
CommandName	String	The command to remove.
ToolbarName	String	The toolbar from which to remove the command. If this is blank, or not included, the command is removed from all toolbars.

Return: There is no return value from this routine.

Method: CommandItem

Description: Retrieves the interface directly to the command item.

For a list of methods and properties available to the CommandItem object, see "[Command Item Object](#)" on page 25.

Object: "Toolbars Object"

Parameters:

Parameter	Type	Description
CommandName	String	The command to retrieve.

Return: This returns a reference directly to the command item.

Method: ConvertImage

Description: Converts a specified image into a file format requested by the client. The `imgfmt` element of the configuration data determines which graphic file formats are available in your system.

See Also: ["imgfmt" on page 584](#)

This method differs from the `SaveAs` method in that it does not depend on the current image. Instead, it lets the client application or script quickly change any file's format.

Object: ["Image Editor Object"](#)

Parameters

Parameter	Type	Description
SrcImagePath	String	The path to the image to convert. If this is empty, the current image is converted and saved.
DestImagePath	String	The location and name to which to save the image. If the file extension specified in this parameter does not match the format parameter, the extension is modified to match the format.
Format	String	The format in which to save the image. See Also: "Specifying Image Format" on page 590
ColorDepth	Long	The depth of the color conversion. See Also: "Specifying Color Depth" on page 590

Remarks

If an image is identified, it is loaded and saved to the given destination in the given format.

See Also: ["Method: SaveAs" on page 117](#)

Return: String - The path to the saved file, with any extension modifications.

If not an empty string, the conversion was a success. If empty, the conversion failed.

See Also: ["Method: ErrorDescription" on page 78](#)

Method: create

Description: Creates an instance of an in-line editor in the page. Returns an `instance object`, which is also added to the `instances` array.

If successful, the editor's name is added to the `eWebEditPro+XML` object to permit easy access to the ActiveX control.

Parameters:

name - Name of the editor. Must match the name of a standard HTML element (typically an input type= hidden) unless the content is to be manually loaded and saved. If the editor is placed on the popup editor page (e.g., ewebeditpropopup.htm), the name is arbitrary.

See Also: [“Appendix A: Naming the eWebEditPro+XML Editor” on page 778](#)

width - The width of the editor in pixels or a percent. For example, 700 or “100%”.

height - The height of the editor in pixels.

If the editor cannot be displayed because **eWebEditPro+XML** is not supported or not installed, a textarea element appears in its place as close in size as possible. Textarea size is specified in rows and columns instead of pixel width and height. You can specify the rows and columns in the parameters object.

parameters (optional) - Optional parameters object for **eWebEditPro+XML**. If not specified, the parameters in the **eWebEditPro+XML** object are used. Parameters supplied to the popup editor take precedence over these.

Object: [“eWebEditPro Object”](#)

Method: createButton

Description: Creates an instance of a button which, if clicked, opens a popup window with the editor in it.

This method must be called even when a custom button is used instead of the standard HTML button. The creation of a standard HTML button may be suppressed by clearing the parameters.buttonTag, in which case, the custom button must call the edit method.

Parameters:

buttonName - The name of the HTML button element to create. The caption that appears on the button is defined as popupButtonCaption in ewebeditpromessages.js. (See [“The ewebeditpromessages File” on page 292.](#))

elementName - The name of the HTML element that stores the content. The element name may contain the form name to differentiate elements of the same name in different forms, for example: “frmMain.Content”.

This name is passed to the edit method when the button is clicked.

parameters (optional) - Optional parameters object for **eWebEditPro+XML**. If not specified, the parameters in the **eWebEditPro+XML** object are used.

Object: [“eWebEditPro Object”](#)

Method: CreateNew

Description: Creates or saves a new image.

Object: [“Image Editor Object”](#)

Parameters

Parameter	Type	Description
Width	Long	Width of new image in pixels
Height	Long	Height of new image in pixels
Depth	Long	Number of colors to give the image <i>See Also:</i> "Specifying Color Depth" on page 590

Remarks

If the current image has been edited but not saved, the user is asked to save the image.

If there is no current image, a new image is created with no prompting.

Return: Boolean - The success of the creation. A true denotes success, a false failure.

Method: DelimitAttributes

Description: This functionality can help a client script process attributes returned from Attributes property. This function formats the values into a set of attributes and values separated by the delimiter value given. This makes parsing the attributes and values easier for the client.

The delimiter used is the pipe (|) character. For example, this attribute set:

attr1="hello" attr2="world"

Is parsed into: attr1|hello|attr2|world

The client then performs a split operation on the string. The client does not need to process the white spaces and equal signs.

See Also: "Method: FormatDelimitedAttributes"; "Delimiting and Un-Delimiting Attributes"

Object: "XML Data Object"

Return: String - The attributes delimited for easy use.

Parameters:

strAttributes	String	The attributes to parse and delimit.
---------------	--------	--------------------------------------

Example:

```

function ShowSetAttributes(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idAttr = 0;
    var strVals = objTag.getPropertyString("Attributes");
    strVals = objTag.DelimitAttributes(strVals);
    if(strVals.length > 0)
    {
        var aryAttrs = strVals.split("|");
        for(idAttr = 0; idAttr < aryAttrs.length; idAttr+=2)
        {
            alert("Attribute " + aryAttrs[idAttr] + " = " + aryAttrs[idAttr+1]);
        }
    }
    else
    {
        alert("There are no attributes for this element.");
    }
}

```

Method: disableStyleSheet

Description: Enables or disables a linked or inline style sheet as identified by its title.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
eWebEditPro.Editor1.disableStyleSheet (strTitle, bDisabled As Boolean)
```

Parameters

strTitle - A unique identifier that represents this style sheet. For an inline style sheet, the title is the tag that the style affects; for a linked style sheet, the title is the style sheet's URL.

bDisabled - Boolean: True: disable the style sheet; (False is not operational as a value)

Example

Assume that you added an inline style sheet.

```
strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")
```

This code disables that style sheet.

```
eWebEditPro.Editor1.disableStyleSheet "P", True
```

This code re-enables it.

```
eWebEditPro.Editor1.disableStyleSheet "P", False
```

Method: disableAllStyleSheets

Description: Enables or disables all style sheets for an editor.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
eWebEditPro.Editor1.disableAllStyleSheets()
```

Method: DocumentTemplate

Description: Uses the active schema to generate a skeleton document template. This template consists of the XML declaration tag, the root tag defined in the schema, and any required elements and attributes under the root element.

Limitations:

- A schema must be loaded and actively selected
- The functionality is only available for XML documents

Object: "XML Object"

Return: string - the template. For example:

```
<?xml version='1.0'?>
<bookcatalog>
    <book id='bk101' visible='true' city='Fresno'>
        <author city='Dallas'> </author>
        <title> </title>
        <description></description>
    </book>
</bookcatalog>
```

Parameters: None

Example:

```
function ShowDocumentTemplate(sEditorName)
{
    var strVal = "";
    var objXml = eWebEditPro.instances[sEditorName].editor.XMLProcessor();

    strVal = objXml.DocumentTemplate();
    alert(strVal);
}
```

Method: DuplicateTag

Description: This method creates a duplicate of a given tag.

When a tag is retrieved through the ActiveTag method, the returned object becomes invalid after the next call to ActiveTag. If the client need to store a tag, it must create a duplicate of that tag.

This function copies all tag information into the new XML Data object. This information can be stored and used later with other attributes and methods, such as ApplyTag.

Object: "XML Object"

Return: CXMLData Object - A copy of the given XML Data object.

Parameters:

Name	Type	Description
objTag	XMLdata	The XML Data object containing the tag data to copy into a new object.

Example:

```
var objNewTag = objXmlDoc.DuplicateTag(objOldTag);
```

Method: edit

Description: Opens a popup window with the editor in it. This method is called when the button created by [createButton](#) is clicked.

Parameters:

elementName - The name of the HTML element that stores the content. The element name may contain the form name to differentiate elements of the same name in different forms, for example: "frmMain.Content".

Object: "eWebEditPro Object"

Method: EditFile

Description: Loads the given file for user editing.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
FilePath	String	<p>The location of the file to edit. This file could be</p> <ul style="list-style-type: none">● on the local system● available on the server● a remote file accessed over the Internet

Remarks

If the file is accessed over the Internet, it should be saved locally. See [Also: "Method: Save" on page 116](#)

Return: Boolean -The success of the load. A true denotes success, a false failure.

Method: EditFromHtml

Description: Parses a specified HTML tag and extracts information about the image and associated named data from the attributes.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
HTML	String	The full HTML tag with all required attributes.

Remarks

The HTML string consists of a fully valid HTML tag. Only one tag is included. Here are some examples:

```
  
  
<table background="\\\\imgserver\\backgrounds\\trees.gif">  
  
<body background="c:\\mystuff\\images\\smooth.gif">
```

All valid attributes and custom attributes are maintained as named data values. The title or alt text is maintained as the description.

For non-IMG tags, the image name is contained in the BACKGROUND attribute. Otherwise, it is contained in the SRC attribute.

Tag	Image FileAttribute	TitleAttribute
body	background	title
img	src	alt
table	background	title
td	background	title

Return: String - the name of the file contained within the tag.

Method: ElementAttributes

Description: Used with the selected schema.

Returns valid attributes to the element delimited with the value specified in the strDelim string value.

You can use this method to display a list of valid attributes that the user can select from. If there are no attributes defined for the element or no schema is selected, the return value is empty.

Object: "XML Data Object"

Return: String - The valid attributes delimited with the character given to the function.

Parameters:

strDelim	String	The delimiter to use between the attribute names.
----------	--------	---

Example:

This example shows all valid attributes for a selected element.

```
function ShowValidAttributes(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var strVals = "";

    strVals = objTag.ElementAttributes(" | ");
    if(strVals.length > 0)
    {
        var aryAttrs = strVals.split(" | ");
        for(idAttr = 0; idAttr < aryAttrs.length; idAttr++)
        {
            alert("Attribute: " + aryAttrs[idAttr]);
        }
    }
    else
    {
        alert("There are no attributes for this element.");
    }
}
```

Method: ElementAttributeValues

Description: Used with the selected schema.

Retrieves the valid values defined in the schema for the given attribute in the current element. The values are delimited using the delimiter value given to the method.

If no valid values are defined, this method returns an empty string.

Object: "XML Data Object"

Return: String - The valid values in a delimited format.

Parameters:

strAttribute	String	The delimiter to use between attribute values.
--------------	--------	--

Example: This example displays the valid values for an attribute.

```
function ShowValidAttributeValues(sEditorName, sAttribute)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idData = 0
    var strVals = "";

    strVals = objElem.ElementAttributeValues(sAttribute, "|");
    if(0 != strVals.length)
    {
        var aryData = strVals.split("|");
        for(idData = 0; idData < aryData.length; idData++)
        {
            alert("Attribute Value: " + aryData[idData]);
        }
    }
    else
    {
        alert("There are no defined values for " + sAttribute);
    }
}
```

Method: ElementChildren

Description: Used with the selected schema.

Retrieves the valid elements defined in the schema for the current element. The values are delimited using the delimiter value given to the method.

If no valid child elements are defined, this method returns an empty string.

Object: "XML Data Object"

Return: String

Parameters:

strAttribute	String	The delimiter to use between element names.
--------------	--------	---

Example:

This example shows all valid elements under a selected element.

```

function ShowValidElements(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idx = 0;
    var strVals = objTag.ElementChildren(" | ");
    if(0 < strVals.length)
    {
        var aryVals = strVals.split(" | ");
        for(idx = 0; idx < aryVals.length; idx++)
        {
            alert("Element: " + aryVals[idx]);
        }
    }
    else
    {
        alert("There are no child elements within '" + objElem.TagName + "'.");
    }
}

```

Method: ElementIsEmpty

Description: Used with the selected schema.

This method returns **true** if the selected element is defined to be empty.

Object: "XML Data Object"

Return: Boolean - **true** if the element is empty.

Parameters: None

Example:

```

If(true == objTag.ElementIsEmpty())
{
    alert("The element does not contain any elements.");
}
else
{
    alert("The element contains child elements.");
}

```

Method: ElementMaxCount

Description: Used with the selected schema.

This returns the maximum times this element can appear within its parent element. A value of -1 means that it is boundless.

Object: "XML Data Object"

Return: Long - The maximum number of times the element can appear.

Parameters: None

Example:

```
alert("Maximum count is " + objTag.ElementMaxCount());
```

Method: ElementMinCount

Description: Used with the selected schema.

This returns the minimum times this element can appear within its parent element. The lowest limit is 0 times, which means that it may not appear.

Object: "XML Data Object"

Return: Long - The minimum number of times the element can appear.

Parameters: None

Example:

```
alert("Minimum count is " + objTag.ElementMinCount());
```

Method: EnableCreation

Description: Enables or disables the user interface that allows the user to create a new image.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
Allow	Boolean	If true , the user can create a new image. The default is true . Even if image creation is not allowed, the client application or script can create a new image. To prevent users from creating new images, make sure the script does not let the user do so through the user interface.

Remarks

One of several methods that control the user interface so that content management systems can operate efficiently and effectively.

Return: Boolean - The setting's previous value so that the caller can restore the value later if needed.

Method: EnableFormatChange

Description: Enables or disables the user's ability to change the file format and select the number of colors for an image.

Object: "Image Editor Object"**Parameters**

Parameter	Type	Description
Allow	Boolean	<p>If true, the user can change the image file's format and color depth. The default is true.</p> <p>See Also: "imgfmt" on page 584; "Specifying Color Depth" on page 590</p> <p>Even if changing an image's format and color depth is not allowed, the client application or script can still change its format or depth.</p> <p>To prevent users from changing the format and color depth, make sure the script does not let the user do so through the user interface.</p>

Remarks

If an image was created, and no format is specified, this setting is ignored.

This is one of several methods that control the user interface so that content management systems can operate efficiently and effectively. For example, changing a file's format may break links to it.

Return: Boolean - The setting's previous value so that the caller can restore the value later if needed.

Method: EnableNameChange

Description: Enables or disables a user's ability to change the name of the image file.

Object: "Image Editor Object"**Parameters**

Parameter	Type	Description
Allow	Boolean	<p>If true, the user can change the image file's name. The default is true.</p> <p>See Also: "namechange" on page 584</p> <p>Even if changing an image's name is not allowed, the client application or script can still change the file name.</p> <p>To prevent users from changing the name, make sure the script does not let the user do so through the user interface.</p>

Remarks

If an image was created and no name is specified, this setting is ignored.

This is one of several methods that control the user interface so that content management systems can operate efficiently and effectively. For example, changing a file's format may break links to it.

Return: Boolean - The setting's previous value so that the caller can restore the value later if needed.

Method: ErrorClear

Description: Clears any current errors.

Object: "[Image Editor Object](#)"

Parameters: None

Remarks

Errors are maintained internally. The client can always retrieve the last error, no matter how far back in the process the error occurred.

This method allows the client to clear errors to ensure that when the user sees an error, it occurred after the error was cleared.

Return: Void

Method: ErrorDescription

Description: Retrieves a text description of the last error encountered.

Object: "[Image Editor Object](#)"

Parameters: None

Remarks

Errors are maintained internally. The client can always retrieve the last error, no matter how far back in the process the error occurred.

There should be an attempt to translate all errors. The return string should be in the language of the user's system.

Return: String - the text description

Method: ErrorValue

Description: Returns a numeric value representing the last error encountered.

Object: "Image Editor Object"

Parameters: None

Remarks

Errors are maintained internally. The client can always retrieve the last error, no matter how far back in the process the error occurred.

This method is used when there is a need to quickly check an error or to avoid the translation issue.

See Also: ["Method: ErrorValue" on page 79](#)

Return: Long - A number value defining the error.

Method: EstimateContentSize

Description: Estimates the size of current content. Use this method with routines that quickly need to know the content size.

The true size is the size of the buffer returned when published content is cleaned and removed.

Parameters:

ContentType - The part of the content to examine. The value can one of these case-insensitive values.

- “whole” - The whole HTML document.
- “body” - The body of the content.
- “text” - The size of the text in the content.

Object: "eWebEditPro Object"

Return: The returned long value is an estimate of the number of characters in the selected content.

Example

Examples of how the EstimateContentSize method can be used in ewep.js.

```
function eWebEditProEditor_save(objValueDestination)
{
    ...
    if(!this.isSizeExceeded(this.editor.EstimateContentSize("WHOLE")))
    {
        this.status = EWEP_STATUS_SAVING;
        var sContent = eval('this.editor.' + this.editorGetMethod + '());
    }
}
```

```

        if (!this.isSizeExceeded(sContent.length))
        {
            objValueDestination.value = sContent;
            this.status = EWEP_STATUS_SAVED;
            this.initEvent("onsave");
            if (this.raiseEvent("onsave") == false)
            {
                return false;
            }
        }
        else
        {
            ShowSizeIsTooLarge(this, "save");
            return false;
        }
    }
    else
    {
        ShowSizeIsTooLarge(this, "save");
        return false;
    }

    . . .
}

function ShowSizeIsTooLarge(objedit, seventsource)
{
    objedit.status = EWEP_STATUS_SIZEEXCEEDED;
    objedit.initEvent("onerror");
    objedit.event.source = seventsource;
    if (objedit.raiseEvent("onerror") != false)
    {
        if (eWebEditProMessages.sizeExceeded)
        {
            alert(eWebEditProMessages.sizeExceeded);
        }
    }
}

```

Method: ExecCommand

Description: Directly executes a command name, with parameters, without going through the **eWebEditPro+XML** command mechanism.

The command is not returned to the client as with the higher level ExecCommand method.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
strCommand	String	A string containing the command, for example, cmdopen
strText	String	A string that may contain text data related to the command. Typically not used.
iData	Long	A long integer value that may contain numeric data related to the command. Typically not used.

Return: None

Method: ExecCommand

Description: Causes the editor to perform the specified operation.

Object: "eWebEditPro ActiveX Control Object"

For more information, see "[Creating a Custom Command](#)" on page 215 and "[Standard Commands](#)" on page 199.

Method: FileExistsLocally

Description: Uses the value given to SrcFileLocationPath to determine if the file exists on the local system.

This can be used for error checking: if the user types in a bad path, this method can detect it.

Object: "Parameters Object"

Return: Boolean

Method: FindDataField

Description: Finds the CXML data object specified by the given xpath. The xpath must start at the root, for example, /root/Group1/Field1.

The xpath can include numeric predicates, for example, /root/Group1[2]/Field1, where Group1 allows more than one. Also, the xpath can be appended with pound sign (#) and number, for example, /root/Group1/Field1#2. This means "select the second field with xpath of /root/Group1/Field1".

NOTE This is *not* standard XPath.

NOTE Predicate and #n are not useful in design mode because fields are not repeated.

Object: "Parameters Object"

Parameters

Parameter	Type	Description
string xpath	String	The xpath to the specified object.

Return: the CXMLData object

Method: FirstCommand

Description: Sets the current reference to the first command available. The reference value held by the script does not change. The reference change is internal to the command mechanism.

To further any enumeration, see "["Method: NextCommand" on page 108](#).

Object: "Command Item Object"

Parameters

Parameter	Type	Description
StrName	String	Receives the name of the first command.
StrCaption	String	Receives the caption of the command. If a text item, it is the text. If a list box, it is the currently selected item text.

Return: If true is returned, it was able to find a command.

Method: Focus

Description: Programmatically sets the focus to the **eWebEditPro+XML** editor using JavaScript. For example:

```
eWebEditPro.instances[sEditorName].editor.focus();
```

Object: "eWebEditPro ActiveX Control Object"

Method: FormatDelimitedAttributes

Description: This method does the opposite of the [DelimitAttributes](#) method. This method is offered to help with client scripting.

When attributes are contained within a delimited string that uses a pipe (|) as a delimiter, this method reformats them as they would appear in an element. The resulting string can be used to assemble an element.

For example, this delimited list: attr1|hello|attr2|world

Is formatted into: attr1="hello" attr2="world"

Object: "XML Data Object"

Return: String - The attribute/value pairs formatted for attributes within an element.

Parameters:

strAttributes	String	The attribute/value pairs delimited with the pipe character.
---------------	--------	--

Example:

```
sAttrVals = objTag.FormatDelimitedAttributes(sDelimAttrs);
```

Method: GetActiveStyleSheetTitles

Description: Returns a comma-delimited list of the titles of the active styles.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
strResult = eWebEditPro.Editor1.getActiveStyleSheetTitles
```

Parameters

strResult - The comma-delimited result set

Example

Given this sequence of adding styles:

```
strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\" & "ektNormal.css")
strResult= eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css")
strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")
strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage3.css")
```

And this disable call:

```
eWebEditPro.Editor1.disableStyleSheet App.Path & "\" & "ektNormal.css", True
```

The call:

```
strResult = eWebEditPro.Editor1.getActiveStyleSheetTitles
```

Yields the three remaining active styles (testpage.css, P, testpage3.css):

```
[value of App.Path]\testpage.css,
P,
[value of App.Path]\testpage3.css
```

Method: getBodyHTML

Description: Saves the content within the BODY tags as HTML. The HTML is a valid fragment.

Object: "eWebEditPro ActiveX Control Object"

Using `getBodyHTML` with `eWebEditPro+XML`

If you are using `eWebEditPro+XML` and a full XML document loads into the editor, `getBodyHTML` returns the full XML document. (That is, its behavior matches the `getDocument` method.) This happens for the following reasons:

- There is no “body” in an XML document.
- Any transformations may prevent the detection of any “body”-like content section.
- `getBodyHTML` is the default method used by the core JavaScript for retrieving content. The method ensures that if you use the default settings in the core JavaScript, the settings work with XML.

If you want to transform an XML document to make it look as well formatted as an HTML document, you must use the internal Load and Save transformation file settings. (An XML document internally transformed into HTML is still recognized as an XML document.)

If you load an XML document that was transformed into an HTML document outside of `eWebEditPro+XML`, the document is considered HTML, *not* XML. In this case, `getBodyHTML` retrieves only the body information.

Example:

```
<p>Both of the buttons below return the same content if a full XML document is loaded.</p>

<input type="button" value="View Full" onClick="window.document.frmeditor1.ViewHTML.value = eWebEditPro.instances['MyContent1'].editor.getDocument()">

<input type="button" value="View Body" onClick="window.document.frmeditor1.ViewHTML.value = eWebEditPro.instances['MyContent1'].editor.getBodyHTML()">
```

Method: `getBodyText`

Description: Returns the content text without formatting. Note that *only* the text is returned, not the html code.

This method is used by browser-based email applications that need both content with HTML tags and content that is text only.

To use this method, first add a hidden field to post the text to the server. Then, when the content is saved, copy the text from the editor into the hidden field.

These steps illustrate how to use this method.

1. Add a hidden field to store the text. For example,
`<input type=hidden name="MyContentText1" value="">`

2. Add JavaScript to copy the text to the hidden field. Use the eWebEditPro.onsave event. This event fires when the content is saved, that is, copied from the editor to the hidden content field.

For example, if formName is the name of your form and MyContent1 is the name of the **eWebEditPro+XML** editor, use this code.

```
<script language="JavaScript1.2">
eWebEditPro.onsave = "document.formName.MyContentText1.value =
eWebEditPro.instances.MyContent1.editor.getBodyText();"
</script>
```

3. Modify your server-side code to process the text. You may wish to save it in a database field for text searches without the HTML tags. Alternatively, you may wish to email the text to clients with text-only viewers.

Object: "eWebEditPro ActiveX Control Object"

Method: GetContent

Description: Retrieves the specified content type from the current edit session. This can be the body of the content, the data entered, or just the header information. Supported content types are listed in "[Content Type Categories](#)" on page 570.

Object: "eWebEditPro ActiveX Control Object"

Parameter: String - the content type to retrieve

Return Type: String - the content retrieved

Example:

```
sContent = objInstance.editor.GetContent("htmlheader");
```

Method: getDocument

Description: Saves the entire HTML document that is currently in the editor.

Object: "eWebEditPro ActiveX Control Object"

Method: GetFieldValue

Description: Reads the value from the given data item. The return value is the value currently assigned to the data item.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
ItemName	String	The name of the data item.

Example

```
txtDataValue.Text = _  
m_objUpload.GetFieldValue(txtDataName.Text)
```

Return

String

Definition of a Field

A field is a named piece of data. When a file is transmitted to the server, fields transmit additional information about the file to the server. Fields consist of a *Field Name* and *Field Data*. The name identifies the field, while the data is the field's contents.

A server can examine the field data and act on the values. An example is a field that transmits the file's category. The server can read this field and, from the category value, determine where to upload the file.

The following is a subset of standard fields. They are normally filled in by the editor when a file is uploaded to the server.

Field	Description
extension_id	A numeric ID that identifies the extension. This can be used to categorize the file in a database. It is offered as a convenience only.
extensions	The list of valid extensions specified in the configuration file. The receiving client can review these extensions to ensure the file being uploaded is acceptable. If the file extension is not acceptable, set the <i>discard</i> attribute of the FILEINFO element to true . <i>See Also:</i> " FILEINFO "
file_size	The file's size in bytes – cannot change.
file_title	The file's description, title, or alt text.
file_type	A numeric value that corresponds to a file type. The value lets a server script determine the type of file being uploaded. The server can then decide how to store and process the file. For a list of file types and their corresponding numeric values, see " Appendix D: Automatic Upload File Types ".
height	The height of the image in the file. If 0, the height is unknown.

Field	Description
img_date	The date of the file – cannot change.
uploadfilephoto	The file selection field – cannot change.
web_media_path	The requested logical location where a browser can find the file, such as <code>http://www.mysite.com/uploads</code> .
width	The width of the image in the file. If 0, the width is unknown.

Method: GetFileDescription

Description: Returns the description of a given file in the list of files added for upload. If the file does not exist in the current list of files, the return value is blank.

Object: "Automatic Upload Object"

Return Type: String

Parameters

Parameter	Type	Description
FileName	String	The full path and name of the file. It cannot be an abbreviated or relative path. The FileName is not case sensitive.

Syntax

```
var sFileDesc = objAutoUpload.GetFileDescription(sUploadFilePathName)
```

Example

```
var objAutoUpload =
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
var sFileDesc = objAutoUpload.GetFileDescription(sUploadFilePathName);
```

Method: GetFileStatus

Description: Retrieves the current upload status of the specified file in the list of files added for upload. The status can be a combination of any values below.

Value	Description
0x00	No activity/doesn't exist in the list of files
0x01	Local file not selected by user for upload
0x02	Local file selected by user for upload.
0x04	Keeping local and not allowing user selection
0x08	Already uploaded
0x10	Local path but doesn't exist locally

If the specified file does not exist in the list, the return value is 0.

Object: "Automatic Upload Object"

Return Type: Long

Parameters

Parameter	Type	Description
FileName	String	The full path and name of the file. It cannot be an abbreviated or relative path. It is not case sensitive.

Syntax

```
ar sFileStat = objAutoUpload.GetFileStatus(sUploadFilePathName);
```

Example

```
var objAutoUpload =
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
var sFileStat = objAutoUpload.GetFileStatus(sUploadFilePathName);
```

Method: getHeadHTML

Description: Returns the <HEAD> through </HEAD> HTML of the current document as a string, including the HEAD tags.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
strHead = eWebEditPro.Editor1.getHeadHTML
```

Example

```
eWebEditPro1.Editor1.getHeadHTML
```

returns

```
<HEAD><TITLE>eWebEditPro Test Page</TITLE>
</HEAD>
```

Method: GetImageInformation

Description: Retrieves specified information about an image.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
InfoName	String	<p>The name of the data item to retrieve:</p> <ul style="list-style-type: none">● width - Image width in pixels. (Not the display width -- the actual width.)● height - The height of the image in pixels.● colors - The color depth, in the format described for colors. (See Also: "Specifying Color Depth" on page 590)● format - The image's format, such as image/gif, image/png, or image/jpg.(See Also: "Specifying Image Format" on page 590)● file name - The saved file name of the image. This is not the assigned name, but the name of the image file saved on the local system. <p>The case is ignored.</p>

Remarks

The method retrieves each item separately to avoid conflicts with structures, collections, or objects that are part of different client applications and scripts.

Here is a VB example:

```
Dim strVal As String
strVal = objEditImage.GetImageInformation("width")
```

After this call, the string value is a number, such as 1280.

Return: String - The value of the requested image property. Numeric values return as a string value representing the number in a decimal format.

Method: getOpenerInstance

Description: Valid in popup pages opened using `eWebEditPro.openDialog()`, this method returns a reference to the Instance JavaScript object responsible for opening this popup.

Example

```
if (eWebEditProUtil.isOpenerAvailable())
{
    var objInstance = eWebEditProUtil.getOpenerInstance();
    var oEditor = objInstance.editor;
    var sSelectedHTML = oEditor.getSelectedHTML();
    :
}
```

Object: "eWebEditProUtil Object"

Method: getProperty

Description: Retrieves the property name given.

This method provides Netscape compatibility.

It is better to use the other `getProperty` methods to return the correct type. If this method is used, the data type is not guaranteed.

Object: "Command Item Object" and "Parameters Object"

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return: The data as a variant. The data type is not guaranteed.

Method: getProperty

Description: Reads from the ActiveX control property.

Object: "eWebEditPro ActiveX Control Object"

Method: getPropertyBoolean

Description: Returns the value of a Boolean property.

Object: "eWebEditPro ActiveX Control Object"

Method: getPropertyInteger

Description: Returns the value of a Numeric property.

Object: "eWebEditPro ActiveX Control Object"

Method: getPropertyString

Description: Returns the value of a String property.

Object: "eWebEditPro ActiveX Control Object"

Method: getPropertyString

Description: Retrieves the property name given as a string.

Object: "Command Item Object" and "Parameters Object"

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return: The data of the property as a string.

Method: getPropertyInteger

Description: Retrieves the property name given as an integer.

Object: "Command Item Object" and "Parameters Object"

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return: The data of the property as an integer.

Method: getPropertyBoolean

Description: Retrieves the property name given as a boolean.

Object: "Command Item Object" and "Parameters Object"

Parameters

Parameter	Type	Description
Name	String	The name of the property to retrieve.

Return: The data of the property as a boolean.

Method: getSelectedHTML

Description: Returns the currently selected content including any HTML tags. The HTML will be a valid fragment.

Pasting the content back into the editor may cause side effects. For example, selecting part of a table returns any HTML tags for a complete table. Pasting it back will insert a table within the table.

Object: "eWebEditPro ActiveX Control Object"

Method: getSelectedText

Description: Returns the currently selected text with no formatting. Only the text is returned, not the html code.

Object: "eWebEditPro ActiveX Control Object"

Method: GetTagAttribute

Description: Retrieves the value of the given attribute within the currently selected tag.

See Also: "Retrieving Attribute Values" on page 674

Object: "XML Data Object"

Return: String - Value of the attribute

Parameters:

strAttr	String	The name of the attribute whose value to retrieve.
---------	--------	--

Example:

```
alert("attr1 = " + objTag.GetTagAttribute("attr1"));
```

Method: GetValidFormats

Description: Retrieves the current set of valid file formats supported by the feature. See Also: "imgfmt" on page 584

Object: "Image Editor Object"

Parameters: None

Remarks

The list of valid formats may not match the list of formats specified with the "SetValidFormats" method. Any formats not supported by the core feature are discarded.

Return: String - the list of valid image formats. See "Specifying Image Format" on page 590.

Method: GetXPath

Description: Returns XPath of selected Data Design field.

See Also: "Method: FindDataField" on page 81

Object: "XML Data Object" on page 23

Parameters: None

Return: Xpath as string

Method: HideAbout

Description: Hides the about command button, if it is shown.

NOTE

It is better to use the `ShowAbout` property, contained within the `eWebEditPro+XML` interface.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
none		

Return: This returns the previous setting for hide.

Method: HideAllMenus

Description: Quickly hides all toolbar menus.

Object: "Toolbars Object"

Parameters: none

Return: There is no return value with this item.

Method: HTMLEncode

Description: HTML encodes the given string.

Example

```
sInputTag += ' value="' + eWebEditProUtil.HTMLEncode(sValue) + '"';
```

Object: "eWebEditProUtil Object"

Method: ImageEditor

Description: Retrieves the Image Edit object that exists within WebImageFX.

The Image Editor object is always returned even if WebImageFX is not installed. It is always best to check with the object to ensure that WebImageFX is available.

Object: "Image Editor Object"

Parameters: None

Example

```
function CheckImageEditor(sEditorName)
{
    var objInstance = eWebEditPro.instances[sEditorName];
    var objImageEdit = objInstance.editor.ImageEditor();
    if(false == objImageEdit.IsPresent())
    {
        alert("The Image Editor is not available.");
    }
}
```

Method: insertMediaFile

Description: Inserts an image file (or other file) to the editor. For images, the IMG tag is used.

This method sets properties in the ActiveX control's Media File Object (see "[The Mediafiles Feature](#)" on page 493) and then executes the cmdmfuinsert command.

Object: "Instances Object"

Parameters: (strSrcFileLocation, bLocalFile, strFileTitle, nWidth, nHeight)

- **strSrcFileLocation** - the path to file being inserted. The path can be the full path or relative to the host name. If a relative path, the editor uses the current page location/BaseURL to determine the file's location.
- **bLocalFile** - **true** if the file is on the user's computer; **false** if the file is on the server.
- **strFileTitle** - the image title; if one is not passed, the user must enter one in the Title field of the Image Selection Screen. It is used as the image's alt text.
- **nWidth** - the width of the image in pixels (if the file is an image)

- **nHeight** - the height of the image in pixels (if the file is an image)

Example

```
eWebEditPro.instances[ "MyContent1" ].insertMediaFile( "mypic.jpg", false, "My Picture Title", 80, 60);
```

Method: IsAttributeRequired

Description: Used with the selected schema.

This returns **true** if the given attribute is defined as required in the selected element.

Object: "XML Data Object"

Return: Boolean - **True** if the attribute is required

Parameters:

strAttribute	String	The attribute name to determine if required.
--------------	--------	--

Example:

```
if(true == objTag.IsAttributeRequired( "attr1" ))
{
    alert("The attr1 attribute is required.");
}
```

Method: isChanged

Description: This method returns

- **true** if the content in any editor on the page was modified
- **false** if no content was changed

See Also: "Method: IsDirty" on page 96

Object: "Instances Object"

Method: isChanged

Description: Use this method to determine if the editor content has changed, for example `eWebEditPro.isChanged()`.

You can also use the **Instance object** method

`eWebEditPro.instances[i].isChanged()`.

This method returns

- **true** if the content in any editor on the page was modified
- **false** if no content was changed

Only editors with modified content have their content copied to the hidden field.

Object: "eWebEditPro Object"

How this Method Emulates onchange

This method enables **eWebEditPro+XML** to emulate the **onchange** event common to standard HTML input elements and the TEXTAREA field. You can combine the **onblur** event with the **isChanged()** method to determine when focus has left the editor and content has been modified.

Method: IsDataField

Description: Returns **true** if the current tag is a Data Design tag.

If the current tag is a custom tag supported by the original XML functionality (for example, <mytag/>), this method returns **false**.

Object: "XML Data Object"

Return: Boolean

Method: IsDirty

Description: Returns a non-zero (boolean true) value if the user has modified the image.

Object: "Image Editor Object"

Parameters: None

Return: Boolean

Method: isEditor

Description: Returns true if an instance of an editor exists by the given name and that instance has a valid 'editor' property.

Return false if an instance of the editor does not exist or does not have an 'editor' property. For example, the instance may be a textarea field because ActiveX is not supported.

Object: "eWebEditPro Object"

Example:

```
if (eWebEditPro.isEditor("MyContent1"))
{
    eWebEditPro.instances[ "MyContent1" ].editor.pasteHTML("Hello World");
}
```

Method: isEditor

Description: Returns **true** if the .editor object is available. Returns false if the .editor object is undefined or null.

Object: "Instances Object"

Example:

```
var objInstance = eWebEditPro.instances[0];
if (objInstance && objInstance.isEditor())
{
    objInstance.editor.pasteHTML("<b>Hello World</b>");
}
```

Method: isEditorReady

Description: If this is **true**, the editor is ready to process a command. If **false**, any commands given or methods called are ignored. This function is normally used only during the "Ready" notification when the editor is loading.

This function is only required when a long series of configuration methods is called in the editor. Because JavaScript is asynchronous, the editor may be processing the previous method when the next JavaScript line is run.

It is good practice to use the time out functionality before checking whether the editor is ready. Often, the next JavaScript line will execute before the editor receives the previous method call.

Object: "eWebEditPro ActiveX Control Object"

The following is an example of using this function.

```

function RunEditorReadyProcess(sEditorName)
{
    // This starts the process of setting up the editor.
    // We need to have timeouts due to the asynchronous nature
    // of JavaScript.  We need to have a wait for each step.

    // We are going to turn off the borders so that the usage looks better.

    eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdshowborders",
    "", 0);

    // The timeout is done before we make the call to
    // check if the editor is ready so that the command
    // can reach the editor and start processing.
    setTimeout('RunPoemTagStep("' + sEditorName + '")', 10);
}

function RunPoemTagStep(sEditorName)
{
    // JavaScript is re-entrant, so the editor may be busy with another
    // script command when this is encountered.  The ready state
    // of the editor should be checked when many commands are run
    // in immediate succession.  This ONLY needs to be checked when many
    // editor commands are run in immediate succession.

    if(eWebEditPro.instances[sEditorName].editor.isEditorReady() == false)
    {
        //Not yet ready, come back later.
        setTimeout('RunPoemTagStep("' + sEditorName + '")', 10);
    }
    else
    {
        RunBasicTempateStep(sEditorName);
    }
}

```

Method: isOpen

Description: Can be used to count the number of open popup windows. A popup window is opened when the user clicks the 'Edit' button created by an **eWebEditPro+XML** function.

This information could be used to alert the user to save and close the popup window prior to submitting.

Example

```
function countOpenPopups()
{
    var iCount = 0;
    for (var i = 0; i < eWebEditPro.popups.length; i++)
    {
        if (eWebEditPro.popups[i].isOpen())
        {
            iCount++;
        }
    }
    return iCount;
}
```

Object: "Popup Object"

Method: isOpenerAvailable

Description: Valid for popup pages, this method determines if the page that opened the popup is still open.

Example

```
if (eWebEditProUtil.isOpenerAvailable())
{
    var objInstance = eWebEditProUtil.getOpenerInstance();
    var oEditor = objInstance.editor;
    var sSelectedHTML = oEditor.getSelectedHTML();
    :
}
```

Object: "eWebEditProUtil Object"

Method: IsPresent

Description: This method returns true if WebImageFX is installed properly on the client's system.

If this method returns **false**, WebImageFX is not installed or is not installed properly. Ektron suggests that a **false** return should disable client scripting functionality that interacts with WebImageFX.

NOTE

If the feature exists on a client but has not been installed properly, this method returns **false**.

Object: "Image Editor Object"

Parameters

None

Method: IsTagApplied

Description: Indicates if a specified XML tag can be applied at the current cursor location.

Object: "eWebEditPro ActiveX Control Object"

Parameters: StrTagName (String) - The number of instances of the custom tag specified with this parameter is counted.

Returns: True if the specified custom tag wraps the current selection. The tag can be *any* tag applied the selection.

If no text is selected, the current cursor location is considered the selection.

Example: You want to verify that a selected style can be applied at the current cursor location. For example, you may want to verify that a tag is being entered at the correct location within your DTD.

```
function ApplyThisTag(sEditorName, strTagInfo)
{
    var objEditor = eWebEditPro.instances[sEditorName];

    if(objEditor.editor.IsTrueApplied("NewsML"))
    {
        eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdcustapplytag", strTagInfo, 0);
    }
    else
    {
        alert("You need to be somewhere within the NewsML section to apply this tag.");
    }
}
```

Method: IsValid

Description: This method offers a quick check to determine if the tag retrieved with ActiveTag is a valid tag.

This method is necessary because a user may select outside of any tag, or the focus may be outside the editor. In these cases, the cursor selection is invalid, and the retrieved tag is an invalid tag.

This does not determine if the tag is valid according to a schema.

Object: "XML Data Object"

Return: Boolean - **true** if a valid selected tag.

Parameters: **none**

Example:

```
if(false == objTag.IsValid())
{
    alert("Please select a tag in the content.");
}
```

Method: IsValid

Description: Returns “true” if the interface references a valid command. If the interface does not reference a valid command, all interface methods and properties are inactive.

The interface may not reference a valid command if you originally set it by referencing a command, and then you delete the command through another interface.

Object: “Command Item Object”

Method: IsVisible

Description: This method returns true if WebImageFX is currently visible to the user from within **eWebEditPro+XML**. A true value means that the user is currently editing an image.

This method returns **False** if WebImageFX is currently not available to the end user. It may not be currently displayed or it may not be installed.

You can use the IsPresent method to determine if the editor is installed on the client system. See Also: “[Method: IsPresent](#)” on page 99.

Object: “Image Editor Object”

Parameters: None

Method: ListCommandName

Description: Available only with list box commands. Returns the name of the command associated with the item at the index specified.

If there is no command associated with that index, it returns an empty string.

Object: “Command Item Object”

Parameters

Parameter	Type	Description
idx	Integer	The 0-based index into the list of commands.

Return: The command name associated with the index. If no command is associated, either the name of the list command or nothing is returned.

NOTE To retrieve the index of the selected list item, use the CommandItem's CmdIndex property: `objCommand.CmdIndex;` or
`objCommand.getPropertyInteger("CmdIndex").`

Method: ListFilesWithStatus

Description: Retrieves a list of files with a specified status.

The list organizes the files and their descriptions in pairs. All values are delimited by the given delimiter value. The file name is the first value, and the description is the second. The first/second list continues for all files.

The editor uses these bit values to designate file status.

Value	Description
0	No activity; will never show in any file retrieval
1	Local file waiting for upload selection
2	Selected by user for upload
4	User selects to keep local
8	Already uploaded to the server
16	Local file but doesn't exist locally
32	File is reserved for later use

When a file is added, it is automatically assigned a status value of 1.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
Status	Long	<p>The or'ed bit value that designates the file's status.</p> <p>Examples of How to Use this Parameter</p> <p>To get a list of files that are either "Local file waiting for upload selection" or "Selected by the user for upload," you can bit wise 'or' the bits together into a number. The files with those statuses are returned.</p> <p>1 or 2 = 3.</p> <p>When a file is uploaded, its status automatically changes to "Already Uploaded." So, to get a list of already uploaded files, specify the "Already Uploaded" bit without or'ing anything with it.</p> <p>To see a list of files that are used but currently local, you could or together the "Local file waiting for upload selection", "Selected by the user for upload," and "User selects to keep local" bits.</p> <p>1 or 2 or 4 = 7</p> <p>Finally, to get a list of every file in the list, regardless of status, set all the bits on. A good shortcut is to use the value -1 because, for PCs, that value sets all the bits on.</p>

Return

String

Example

```

function ListFilesWithStatus(iSelectStat)
{
    if((iSelectStat >= 0) && (iSelectStat < 8))
    {
        var objAutoUpload = GetAutoUploadObject();
        if((null != objAutoUpload) && ("undefined" != typeof objAutoUpload))
        {
            var sList =
                objAutoUpload.ListFilesWithStatus(g_iFileStatusList[iSelectStat], " | ");
            if(sList.length > 0)
            {
                var aryQuery = sList.split(" | ");
                var pair = [];

                for(var i = 0; i < aryQuery.length; i+=2)
                {
                    alert(aryQuery[i+1] + " [" + aryQuery[i] + "]");
                }
            }
            else
            {
                alert("No files came back with that status.");
            }
        }
        else
        {
            alert("Could not get an Auto-Upload object. Can't list files.");
        }
    }
    else
    {
        alert("Invalid status of '" + iSelectStat + "' -- can't list files.");
    }
}

```

Method: ListLength

Description: Returns the number of items in the list. See *Also:* ["Specifying Items in a Select List Field"](#)

Object: "XML Data Object"

Parameters:

none

Example

JavaScript example

```
var numItems = objDDField.ListLength();
```

Method: load

Description: Loads content into editor. Not typically needed. valueSource may be

- undefined (content is loaded from the content element)
- an object with a 'value' property
- a string

Object: "Instances Object"

Method: load

Description: Loads content into all the in-line editors on the page from the standard HTML elements (typically an `input type=hidden` field) with the same name.

Object: "eWebEditPro Object"

Method: LoadedFileName

Description: Returns the name of the loaded image file. This is the original name, not the one selected by the user. This is normally the remote location of the image, but it can be an image loaded from the local system.

This can be used as a key to the image.

This will match the value returned by SavedFileName if a local file was loaded.

Object: "Image Editor Object"

Parameters: none

Return: string

Method: LoadList

Description: Creates the list of items from an XML document. See Also:
["Specifying Items in a Select List Field"](#)

Object: "XML Data Object"

Parameters:

Parameter	Description
strSource	An XML document, either string or URL, used to create the list
strSelect	An XPath expression identifying the element that contains the caption and value

Parameter	Description
strCaptionXPath	An XPath expression relative to 'strSelect' that specifies the caption value
strValueXPath	An XPath expression relative to 'strSelect' that specifies the value itself
strNamespace	Any namespaces used in the XPath expressions

Return: error code. 0=no error.

Example:

Javascript example

```
objDDField.LoadList("/employeelist.asp", "//employee", "name", "@empid","");

```

Method: Locale

Description: Specifies the locale file to use. If this method is not set, the locale is determined by system settings. If this is set, the locale file specified is used.

See Also: ["Displaying Menus and Dialogs in a non-European Language"](#)

Object: ["Parameters Object"](#)

Parameters: none

Return: string

Example:

```
<input type=hidden name="MyContent1" value">
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{
    eWebEditPro.parameters.locale = "locale0000b.xml";
    eWebEditPro.create("MyContent1", "100%", 400);
}
//-->
</script>
```

Method: MediaFile

Description: Returns a reference to the Media File object. All media file functionality is accessed through this object.

See Also: ["Media File Object"](#) on page 514

Object: ["eWebEditPro ActiveX Control Object"](#)

Method: MinChildElementCount

Description: Used with the selected schema.

This method returns the minimum number of times a given child element can exist under the current element. A value of -1 means there is no limit to how many times it can exist.

If the value returned is 0, it is optional for the element to appear.

If the value returned is other than 0, the element must appear.

Object: "XML Data Object"

Return: Long - The number of times the given element can exist under the current element.

Parameters:

strElement	String	The name of the child element that is under the current element. Case matters.
------------	--------	--

Example:

This example goes through all valid elements and marks the required ones.

```
function ShowValidElementsWithRequired(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idx = 0;
    var strVals = "";

    strVals = objTag.ElementChildren(" | ");
    if(0 < strVals.length)
    {
        var aryVals = strVals.split(" | ");
        for(idx = 0; idx < aryVals.length; idx++)
        {
            if(objElem.MinChildElementCount(aryVals[idx]) > 0)
            {
                alert("Element: " + aryVals[idx] + " is required.");
            }
            else
            {
                alert("Element: " + aryVals[idx]);
            }
        }
    }
    else
    {
        alert("There are no child elements within '" + objElem.TagName + "' .");
    }
}
```

Method: NextCommand

Description: Sets the current reference to the next command available. The reference value held by the script does not change. The reference change is internal to the command mechanism.

To initiate any enumeration, see “[Method: FirstCommand](#)” on page 82.

Object: “[Command Item Object](#)”

Parameters

Parameter	Type	Description
StrName	String	Receives the name of the first command.
StrCaption	String	Receives the caption of the command. If a text item, it is the text. If a list box, it is the currently selected item text.

Return: If true is returned, the method found a command. If it returns false, there are no more commands to enumerate. The reference will be on the last command enumerated.

Method: openDialog

Description: Opens the popup Web page specified by fileName. The given editor name is defined as 'editorName' in the URL query string parameter.

In the popup page, include eweutil.js and then use eWebEditProUtil.editorName to retrieve the editor name.

See Also: “[eWebEditProUtil Object](#)” on page 4

You can also specify the window name and window features. The window name and window features are parameters to the standard window.open() JavaScript method.

Object: “[eWebEditPro Object](#)”

Parameters: editorName, fileName, query, windowName, windowFeatures

Example:

```
function showFormElementDialog(sEditorName, sFormElement, sWin, width, height)
{
    var sWindowFeatures = "scrollbars,resizable,width=" + width + ",height=" + height;
    var sFilename = "formelementinsert.htm";
    eWebEditPro.openDialog(sEditorName, sFilename, "formelement=" + escape(sFormElement),
sWin, sWindowFeatures);
}
```

Method: outerXML

Description: Returns the XML of the custom tag as a string, for example, <mytag>some text</mytag>.

Method: openDialog

Description: Opens the popup Web page specified by fileName. The given editor name is defined as 'editorName' in the URL query string parameter.

In the popup page, include eweutil.js and then use eWebEditProUtil.editorName to retrieve the editor name.

See Also: "eWebEditProUtil Object" on page 4

You can also specify the window name and window features. The window name and window features are parameters to the standard window.open() JavaScript method.

Object: "eWebEditPro Object"

Parameters: editorName, fileName, query, windowName, windowFeatures

Example:

Method: outerXML

Description: Returns the XML of the custom tag as a string, for example, <mytag>some text</mytag>.

Object: "XML Data Object"

Method: pasteHTML

Description: replaces the selected **eWebEditPro+XML** content with the string passed to pasteHTML.

sHTMLText: the string pasted into the content at the current cursor location when pasteHTML is executed. This string replaces any selected content. For example

```
eWebEditPro.Editor1.pasteHTML( "<hr><br><b>Hello World!</b>" );
```

sHTML text can be plain text (for example, "hello world") or HTML (for example, "Hello <i>World!</i>").

The following example pastes HTML from a text field (Text1) into an editor named MyContent1 when the Paste button is pressed.

```
<input type="text" name="Text1" value="<i>paste</i> <b>this</b>">
<input type="button" name="btnPaste1" value="Paste"
onclick="eWebEditPro.instances.MyContent1.editor.pasteHTML(Text1.value)">
```

For a complete sample, see the **eWebEditPro+XML** sample page, ewebeditpro.htm.

Object: "eWebEditPro ActiveX Control Object"

Method: pasteText

Description: Replaces selected content in **eWebEditPro+XML** with the string passed to `pasteText`. The content is pasted as is. HTML tags are not interpreted.

sText: the content to be pasted into the editor's content at the current cursor location. Any editor content that is selected when `pasteText` is executed is replaced.

For example:

```
eWebEditPro.Editor1.pasteText( "Hello World! " );
```

Object: "eWebEditPro ActiveX Control Object"

Method: PopulateTagsWithStyles

Description: Applies the current, active styles to the content's tags.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
bResult = eWebEditPro.Editor1.populateTagsWithStyles
```

Parameters

`bResult` - Boolean True: Success; False: Failure

Defaults

This function adds some harmless default values: note "BOTTOM" and "FILTER" in the example below.

Precedence

When rendering content, styles embedded in content tags take precedence over header style tags.

Example

Given the style sheet added inline:

```
strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family:Arial")
```

Where the resulting style is:

```
<STYLE disabled title=P>P {  
    FONT-FAMILY: Arial  
}
```

And content represented by this HTML:

```
<P>Sentence one</P>  
<P>Sentence two</P>  
<P>&nbsp;</P>
```

Calling `PopulateTagsWithStyles` yields:

```
<P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">Sentence one</P>
```

```
<P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">Sentence two</P>
<P style="BOTTOM: 0px; FILTER: ; FONT-FAMILY: Arial">&nbsp;</P>
```

Method: PopupMenu

Description: Brings up a popup menu.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
MenuName	String	The name of the Popup Menu to bring up. If the menu does not exist or is not a popup menu style, nothing happens
RelativeCmd	String	The command associated with the popup. Optional

Return: There is no return value.

Method: PublishHTML

Description: Takes the named values and formats them into an HTML tag that contains attribute/value combinations.

Object: "Image Editor Object"

Parameters: None

Remarks

The HTML string consists of a fully valid HTML tag. Only one tag is included. Here are some examples:

```


<table background="\\\\imgserver\\backgrounds\\trees.gif">

<body background="c:\\mystuff\\images\\smooth.gif">
```

All valid attributes and custom attributes are maintained as named data values. The title or alt text is maintained as the description.

For non-IMG tags, the image name is contained in the BACKGROUND attribute. Otherwise, it is contained in the SRC attribute.

Tag	Image FileAttribute	TitleAttribute
body	background	title

Tag	Image FileAttribute	TitleAttribute
img	src	alt
table	background	title
td	background	title

Return: String - The correctly formatted HTML that contains the information about the image

Method: ReadNamedData

Description: Retrieves the data value of the data name from the file specified.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
filename	string	The filename in the file store where the named data set is located.
data name	string	The name/id of the named data set.

Example

```
example: sDValue = objAuto.ReadNamedData(sFileName, sDName);  
  
or  
sDValue = objAuto.ReadNamedData("c:\abc.jpg", "id");
```

Return: string

Method: ReadResponseHeader

Description: Retrieves the header of the response sent by the server.

Object: "Automatic Upload Object"

Parameters: None

Example

```
function ShowResponseHeader()  
{  
    var objAutoUpload = GetAutoUploadObject();  
    if((null != objAutoUpload) && ("undefined" != typeof  
objAutoUpload))
```

```

        {
            window.document.frmEditor1.ServerResponse.value =
                objAutoUpload.ReadResponseHeader();
        }
        else
        {
            alert("Could not get an Auto-Upload object.");
        }

        alert("Completed retrieving the response header.");
    }

```

Return: String

Method: ReadUploadResponse

Description: Reads the full text returned from the server as a response to the upload. The return value is normally an HTML page or XML data.

Object: "Automatic Upload Object"

Parameters: None

Example

```
txtResponse.Text = m_objUpload.ReadUploadResponse
```

Return: String

Method: refreshStatus

Description: Updates the values of the following properties:

- status
- isIE
- isNetscape
- browserVersion
- isSupported
- isAutoInstallSupported
- isInstalled
- versionInstalled
- upgradeNeeded

Object: "eWebEditPro Object"

Method: relocate

Description: frameName = name of the frame that includes ewebeditpro.js

This method relocates the 'on' event handlers to point to the frame where the functions are actually defined. The frame that includes ewebeditpro.js is the frame that defines the event handler functions.

For example

```
var eWebEditPro = top.eWebEditPro;  
eWebEditPro.parameters.relocate("top");
```

This method is typically not required.

Object: "Parameters Object"

Method: RemoveFieldValue

Description: Removes the given data item so it is not sent with the upload. When a data item is removed, it is no longer sent with the file upload.

IMPORTANT!

Be careful! Standard fields can be removed, just as they can be changed, and it may be necessary to remove them. However, if a standard field is removed, undesired consequences may result.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
ItemName	String	The name of the data item

Example

```
m_objUpload.RemoveFieldValue txtDataName.Text
```

Return: None

Method: RemoveFileForUpload

Description: Removes a specified file from the list of files for uploading.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
LocalFileName	String	The name and path of the local file to upload.

Example

```
function RemoveSelectedFile(sFileName)
```

```

{
    var objAutoUpload = GetAutoUploadObject();
    if((null != objAutoUpload) && ("undefined" != typeof objAutoUpload))
    {
        objAutoUpload.RemoveFileForUpload(sFileName);
    }
    else
    {
        alert("Could not get an Auto-Upload object. Can't list files.");
    }
}

```

Return

None

Method: RemoveListItem

Description: Removes an item, specified by `Index`, from the list. The `Index` is zero-based. That is, the first item in the list is index 0. *See Also:* "Specifying Items in a Select List Field"

Object: "XML Data Object"

Parameters:

none

Return: error code. 0=no error.

Example

JavaScript example

```
objDDField.RemoveListItem(0);
```

Method: RemoveNamedData

Description: Removes the named data set from the file specified.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
filename	string	The filename in the file store where the named data set is located.
data name	string	The name/id of the named data set.

Example

```
objAuto.RemoveNamedData(sFileName, sDName);
```

or

```
objAuto.RemoveNamedData("c:\abc.jpg", "id");
```

Return: boolean

Method: reset

Description: Reinitializes all values to the default defined in eWebEditProDefaults (ewebeditprodefaults.js). This method should be called after creating an editor if properties were changed for that instance of the editor.

If reset() is not called, any changed property values apply to all subsequent instances of the editor.

Object: "Parameters Object"

Method: resolvePath

Description: Prepends the URL with the eWebEditPro path (for example, /ewebeditpro5/).

Object: "eWebEditPro Object"

Method: RetrieveHTMLString

Description: Returns the HTML string that will be used for insertion into HTML.

Object: "Parameters Object"

Return: Boolean

Parameter: bAsIs (Boolean) Keep the name and path as assigned. Do not modify to use the remote path or resolve the reference path.

A value of **True** usually means use the local path and not the remote path.

Return: The HTML string that would be inserted into the document.

Method: save

Description: Saves content. Not typically needed. objValueDestination may be

- undefined (content is stored to the content element)
- an object (the value property will be set)

Object: "Instances Object"

Method: Save

Description: Saves the currently edited image with the currently selected file parameters.

Object: "Image Editor Object"

Parameters: None

Remarks

If parameters are missing, such as the file name, the user is prompted to supply those values.

Return: String - the full file name of the saved image. An empty string denotes an error.

Method: save

Description: Saves content from all the in-line editors to the standard HTML elements (typically an `input type=hidden` field) with the same name.

Object: "eWebEditPro Object"

Method: SaveAs

Description: Saves the currently edited image with the specified parameters.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
FileName	String	The name of a file that will hereafter exist on the local system. Remote Internet addresses are not allowed.

Remarks

To change the format as well as the file name, see "["Method: ConvertImage" on page 65](#).

Return: String - the full file name of the saved image. An empty string denotes an error.

Method: SavedFileName

Description: Returns the name that the file was actually saved as. Since this is quite often a temp name, or one chosen by the user, the client can't depend on the save name matching the loaded file name.

This will match the value returned by LoadedFileName if a local file was loaded.

Object: "Image Editor Object"

Parameters: None

Return: String

Method: SchemaExists

Description: This method allows the client to check if the schema assigned the given namespace is loaded. The namespace is URI assigned to the schema when it is loaded. See [AddSchema](#) to learn how namespaces are assigned to schemas.

See Also: "Working with Schemas" on page 666

Object: "XML Object"

Return: Boolean - **True** if the given namespace has a loaded schema.

Parameters: None

Example:

```
If(true == objXmlDoc.SchemaExists(sSchemaName))  
{  
    alert("The schema exists.");  
}
```

Method: SeparatorBarAdd

Description: Adds a separator bar to the specified toolbar. On a toolbar, it is a vertical bar. On a popup menu, it is a horizontal bar. It is mostly used to organize commands into groups.

See Also: "Adding a Separator Bar Between Two Toolbar Menu Items"

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
CommandName	String	The separator bar is assigned an internal name. This value receives that name. It is used as a reference if it is modified.
ToolbarName	String	The name of the toolbar or menu to which to add the bar.
iPosition	Integer	Position of the command within the given toolbar. If omitted or - 1, it is placed at the end.

Return: This returns true if it successfully created the separator.

Method: SeparatorSpaceAdd

Description: Adds a separator space to the specified toolbar. It is used mainly to organize commands into groups.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
CommandName	String	The separator space is assigned an internal name. This value receives that name. It is used as a reference if it is modified.
ToolbarName	String	The name of the toolbar or menu to which to add the bar.
iPosition	Integer	Position of the command within the given toolbar. If omitted or - 1, it is placed at the end.

Return: This returns true if it successfully created the separator.

Method: setBodyHTML

Description: This method does not exist. To load HTML content into the editor, use the setDocument() method.

Object: "eWebEditPro ActiveX Control Object"

Method: SetConfig

Description: Specifies which configuration file to use for controlling WeblImageFX. This can be either a local file, a remote file, or an XML data stream.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
Configuration	String	The location of the configuration or the stream of XML characters defining the configuration.

Method: SetContent

Description: Assigns the given content to the editor session. Supported content types are listed in ["Content Type Categories" on page 570](#).

If a content type requires the editor be in a special mode, such as Data Design, XML, HTML mode, the editor switches into the mode that allows the content to be processed.

Object: "eWebEditPro ActiveX Control Object"

Parameters:

Parameter	Type	Description
Type	String	The type of content being set.
Content	String	The content to place into the current session.
Data	String	If this is not an empty string, it is a string of data to associate with the given content.

Return Value: Boolean - True if successful

Example:

```
objInstance.editor.SetContent("htmlbody", strBody, "");
```

NOTE

If the editor and the field have the same name, but you want to set the value of the editor content to something other than the default text, the value that you set does not get displayed. This is because the editor and field names are the same, so the default text of the editor takes precedence over the set value. To change the default content, assign unique names to the editor and field.

Method: setDocument

Description: Replaces the entire document, including all tags outside of the body tag and style information, with the specified document. Any previous document is completely lost.

Object: "eWebEditPro ActiveX Control Object"

Parameter

strDoc - String - The HTML document to place into the editor. This must be a complete and valid document that contains the doctype, html, head, and any other tags required for correct display of the document.

Return: Nothing

Example 1:

```
function SetFullDocument()
{
    var objEdit = eWebEditPro.instances.MyContent1.editor;
    objEdit.setDocument(DocHTML.value);
}
```

Example 2:

```
<input type="button" value="Set Document"
onClick="eWebEditPro.instances.MyContent1.editor.setDocument(DocHTML.value)">
```

Method: SetFieldValue

Description: Adds or modifies a field which is posted with either the content or file. This value is received by the server as if it were a text field on a form.

The name given is the name of the posted field. The value is the string data to place in that field. The receiving server extracts the value as it would any posted text field.

If the field already exists, the given value replaces the data. If a script must append, it needs to read, append, then write the data.

See Also: ["Definition of a Field"](#)

Object: ["Automatic Upload Object"](#)

Parameters

Parameter	Type	Description
FieldName	String	The name of the data item.
Value	String	The value to assign to the data item. This can be a blank string.

Example

```
var objAutoUpload =
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
objAutoUpload.SetFieldValue(sField, sDataValue);
```

Return

None

Method: SetFileDescription

Description: Sets the description of the specified file. This description is posted to the server with the file. Each file has its own description posting.

If the file does not exist, the file is added with the given description.

Object: ["Automatic Upload Object"](#)

Parameters

Parameter	Type	Description
FileName	String	The full path and name of the file. It cannot be an abbreviated or relative path. It is not case sensitive.
Description	String	The description to post with the file.

Example

```
var objAutoUpload =
  eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
objAutoUpload.SetFileDescription(sUploadFilePath, sDescription);
```

Return

None

Method: SetFileStatus

Description: Sets the status of the given file. This allows a script to select or unselect a file for upload. (The user must still approve any upload process.)

The status value can be a combination of any of the values below.

Value	Description
0x00	No activity/doesn't exist in the list of files
0x01	Local file not selected by user for upload
0x02	Local file selected by user for upload.
0x04	Keeping local and not allowing user selection
0x08	Already uploaded
0x10	Local path but doesn't exist locally

If the file does not exist, no action is taken.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
FileName	String	The full path and name of the file. It cannot be an abbreviated or relative path. It is not case sensitive.
Description	String	The description to post with the file.

Example

```
var objAutoUpload =  
eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();  
objAutoUpload.SetFileStatus(sUploadFilePath, 0x01);
```

Return

None

Method: setHeadHTML

Description: Sets the <HEAD> through </HEAD> portion of the document header.

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
eWebEditPro.Editor1.setHeadHTML(strReplacementHead)
```

Parameter

strReplacementHead - The HTML <HEAD>...</HEAD> replacement string.

Remarks

WARNING! Do not add styles using this method. They are not supported, and the header will reflect incorrect information.

This feature replaces all header information. If the new header information includes styles, the style information will appear in the HEAD tag area, but will not remove, replace or add any style information.

Example

This replaces the header with just a TITLE element:

```
eWebEditPro.Editor1.setHeadHTML "<HEAD<TITLE>New Header</TITLE></HEAD>"
```

Method: SetLocale

Description: Specifies a Locale translation file to use. This can be a local file, a remote file, or an XML data stream.

See Also: "Modifying the Language of eWebEditPro+XML" on page 265

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
Locale	String	The location of the localization data or the stream of XML characters defining the localization data.

Method: setProperty

Description: Sets the named property to the value given.

See the getProperty series of methods (beginning with "["Method: getProperty"](#)") on how to retrieve values.

Object: "Command Item Object"

Parameters

Parameter	Type	Description
Name	String	The name of the property.
Value	Variant	The data to set into the property.

Return: Nothing

Method: setProperty

Description: Writes to the ActiveX control property.

NOTE This property is intended for environments such as Netscape, which do not directly support properties.

Object: "[eWebEditPro ActiveX Control Object](#)" and "[Parameters Object](#)"

Example

```
function SetAutoUploadProperty(sVarName, sVarValue)
{
    var objAutoUpload = GetAutoUploadObject();
    if((null != objAutoUpload) && ("undefined" != typeof objAutoUpload))
    {
        objAutoUpload.setProperty(sVarName, sVarValue);
    }
}
```

```

        else
    {
        alert("Could not get an Auto-Upload object.");
    }
}

```

Method: SetTagAttribute

Description: Sets the given attribute value to the value sent to the method.

The change is reflected once the XML Tag object is applied using the ApplyTag method in the XML Object Interface.

Object: "XML Data Object"

Return: Boolean - **True** if the attribute could be set in the element.

Parameters:

strAttr	String	The name of the attribute. Case matters
strVal	String	The value to assign to the attribute.

Example:

```
objTag.SetTagAttribute("attr1", "aloha");
```

Method: SetValidFormats

Description: Specifies a set of formats that are considered valid by a client application or script.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ValidFormats	String	The list of valid formats. See " Specifying Image Format " on page 590 to learn about image file formats

Remarks

If a format is not supported by WebImageFX, that format is discarded. If the number of supported formats is 0, an error is generated.

Return: Long - The number of formats now supported

Method: ShowAbout

Description: Shows the about button if defined in the XML data.

NOTE

It is better to use the [ShowAbout property](#), contained within the [eWebEditPro+XML interface](#).

Object: "Toolbars Object"

Parameters

none

Return: This returns the previous show state of the about button.

Method: ShowActiveStylesDetails

Description: Returns a comma-delimited list of the active style sheet titles and style information (CSS syntax text).

If all of a style's rules are overridden but the style is still active (that is, not "disabled"), the value of that style returns the phrase: "No Active Rules".

Object: "eWebEditPro ActiveX Control Object"

Syntax

```
strStyles = eWebEditPro.Editor1.showActiveStylesDetails
```

Parameters

(result) - The comma-delimited list of style sheet titles and their values

Example

Given adding these styles:

```
strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage.css")
strResult = eWebEditPro.Editor1.addLinkedStyleSheet(App.Path & "\testpage3.css")
strResult = eWebEditPro.Editor1.addInlineStyle("P", "font-family: ""lucida console""")
```

Where:

testpage.css defines a style for the P tag

testpage3.css defines styles for P, H1 and H2

The third line inserts an inline P tag style

So that:

testpage.css adds a P style

testpage3.css overrides that P style and adds a style for H1 and H2

the inline style overrides the P style yet again

Calling `eWebEditPro.Editor1.showActiveStylesDetails` yields the following results.

Stylesheet:

```
C:\EKTRON~1\DEVELO~1\EWEBED~1\v2\Test\TestApp\testpage.css, No Active Rules
```

Stylesheet:

```
C:\EKTRON~1\DEVELO~1\EWEBED~1\v2\Test\TestApp\testpage3.css, cssText:  
H1 {FONT-FAMILY: "Arial"; FONT-SIZE: 11pt; MARGIN: 0in}  
H2 {FONT-FAMILY: "Arial"; FONT-SIZE: 10pt; MARGIN: 0in}
```

Stylesheet:

```
P, cssText: P {FONT-FAMILY: "lucida console"
```

Method: ShowAllMenus

Description: Restores the view of menus hidden with "["Method: HideAllMenus"](#)".

Object: "["Toolbars Object"](#)"

Parameters

Parameter	Type	Description
none		

Return: There is no return value.

Method: TagCount

Description: Indicates how many times a specified XML tag exists in the content.

Object: "["eWebEditPro ActiveX Control Object"](#)"

Parameters: StrTagName (String) - The name of the custom tag to search for and count the occurrences of.

Returns: The number of times a custom tag is used in the content. This is a long integer value.

Example: Before loading content, you want to check and see if it has already been loaded. To check for this, use this code.

```
if(objEditor.editor.TagCount( "NewsML" ) == 0)  
{  
    objEditor.editor.pasteHTML( "content goes here" );  
}  
else  
{  
    alert("You already have a news item in your content.");  
}
```

Method: Thumbnail

Description: Creates a thumbnail of the current image or a specified image file.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ImageFile	String	The location of a file to load and from which to create a thumbnail. If this value is empty, a thumbnail is made of the current image. If there is no current image and no ImageFile location, an error occurs.
DestFile	String	The thumbnail's destination location. If this value is empty, an error occurs.
Width	Long	The thumbnail's width in pixels. If the value is 0, the width maintains proportionality with the height. If both width and height are 0, the width defaults to 32 and the height maintains proportionality with the width.
Height	Long	The height in pixels. If the value is 0, the height maintains proportionality with the width.
Colors	Long	The bit depth. If no value is specified, the bit depth is 8 (256 colors). See Also: " "Specifying Color Depth" on page 590
Format	String	The image file format. If blank, the format is determined by the extension of the current or loaded file. See Also: " "Specifying Image Format" on page 590

Remarks

If a current image has unsaved edits and it must be replaced with a specified image (via the `ImageFile` parameter), the user is asked to save the modified image.

If information is missing, the user is prompted to supply it.

Return: String - The resulting full file name. This may not match the given file name because the specified format extension may not match the given filename extension.

Method: ToolbarAdd

Description: Creates a toolbar and adds it to the toolbars available to the user.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
ToolbarName	String	The name of the toolbar. This must be unique among all currently created toolbars.
Caption	String	The toolbar caption.
CaptionAlignment	etbCaptionAlignment	The alignment of the toolbar caption. <i>See Also:</i> "etbCaptionAlignment" on page 260
Style	etbToolbarStyles	The style of the toolbar. <i>See Also:</i> "etbToolbarStyles" on page 260
Options	Long	Bit field of etbToolbarOptions bits describing specific options for the toolbar. <i>See Also:</i> "etbToolbarOptions"
Position	etbToolbarLocation	Toolbar position. <i>See Also:</i> "etbToolbarLocation"
ParentMenu	String	The name of the parent menu. This is for use with sub-menus. Optional.

Return:

Returns an etbErrorValues value.

Method: ToolbarModify

Description: Modifies an existing toolbar.

Object: "Toolbars Object"

Parameters

Parameter	Type	Description
-----------	------	-------------

ToolbarName	String	The name of the toolbar to change.
Modification	etbToolbarModification	How to modify the toolbar. See Also: "etbToolbarModifications" on page 261

Return: Returns an etbErrorValues value.

Method: Toolbars

Description: Returns a reference to the Toolbar Interface object. All toolbar functionality and toolbar interfaces are accessed through this interface.

See Also: "Toolbars Object" on page 19; "The Toolbar Object Interface" on page 258

Object: "eWebEditPro ActiveX Control Object"

Method: Transform

Description: Performs a transformation on the document content given to the method. The XSLT must be a file accessible to the instance of the editor.

The string returned is the content modified by the XSLT.

NOTE Ektron's eWebEditPro+XML+XML Knowledge Base contains several articles that describe using XSLTs. To read the articles, go to <http://www.ektron.com/support/ewebeditprokb.cfm> and enter XSLT into the Search field.

Object: "XML Object"

Return: String - The modified content

Parameters:

Name	Type	Description
strXMLDocument	String	The document content to transform. This is normally a whole XML document. It can be the XML document retrieved from the editor and then transformed for different display environments.
strXSLT	String	The location of an XSLT file. This must be a file and not a stream. It must be accessible to the instance of the editor. This can be a relative or fully qualified path.

Example: This example shows how to perform a transformation on content.

```

function TransformAndShowContent(sEditorName, xContentString, sXSLTFile)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var sTransformed = objXmlDoc.Transform(sHtmlString, sXsltPath);
    if(objXmlDoc.getPropertyInteger("ErrorCode") == 0)
    {
        eWebEditPro.instances.MyContent1.editor.setDocument(document.frmeditor1.editorinput.value);
    }
    else
    {
        ShowFormattedContent(sTransformed);
    }
}

```

Method: UploadConfirmMsg

Description: Sets the user message displayed on the user intervention dialog. This dialog is required for security. The user must perform an action before an upload is allowed.

The message specified is shown to the user. There must be two possible answers to this message:

- Yes - the upload will proceed
- No - the upload will not proceed

For example, the message can indicate the proposed upload location. The user can decide if he or she wants to place the file in that location. The user can select **No** and, instead, use the server's external mechanism to select a category or location.

Object: "Automatic Upload Object"

Parameters

Parameter	Type	Description
YesNoQuestion	String	The question asking the user whether or not they want to proceed with the upload.
Title	String	The title of the dialog

Example

```
m_objUpload.UploadConfirmMsg strQuestion, strTitle
```

Return: None

Method: UseHTMLString

Description: The information from the given HTML string is extracted and placed into the appropriate Media object properties. This includes the file name, size, position, and other values that can be specified in HTML.

Object: "Parameters Object"

Parameter: StrHTML (String) The HTML string to extract values from.

Return: None

Method: Validate

Description: This method validates a given set of content. The content can be from the editor or another source. The content can also have been transformed so that it is formatted correctly for the assigned schema.

The schema does not need to have been loaded before this call. If the schema is not loaded, it is loaded before validation occurs.

If the schema was loaded before this call, the strSchema value can be empty and the strNameSpace value is used to select the validation schema.

If a schema file is given and the namespace value is empty, there is an attempt to locate the schema in the list of loaded schemas. If the schema is not loaded, it is loaded using either the path or the internal namespace as the URI.

If both the schema and namespace are blank, no schema is selected for validation. The validation information must exist in the XML data.

If a DTD is used for validation, both strSchema and strNameSpace values must be empty, and the DTD must be specified in the XML data.

Use the Error properties to determine any errors. If ErrorCode has a value of 0, no validation errors.

See Also: "[Validating XML Content](#)" on page 663

Object: "XML Object"

Return: Boolean - **True** if the validation succeeds.

Parameters:

Name	Type	Description
strXMLDocument	String	The content to validate. This is typically a complete document.
strSchema	String	The schema file to use for validation. Set as empty if the schema is already loaded, or a DTD is being used, or the schema is specified in the XML content.
strNameSpace	String	The namespace of the schema. Set blank if you want to load the given schema and have an assigned namespace used, or a DTD is used for validation, or the information is contained in the XML content.

Example: This example takes content and validates it.

```
function ValidateXML(sEditorName, sXMLString, sSchemaPath, sNSTarget)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    objXmlDoc.Validate(sXMLString, sSchemaPath, sNSTarget);
    if(objXmlDoc.getPropertyInteger("ErrorCode") == 0)
    {
        alert("Passed!");
    }
    else
    {
        alert("ERROR: " + objXmlDoc.getPropertyInteger("ErrorCode") + " : " +
        objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

Method: ValidChildElement

Description: Used with the selected schema.

This method offers a quick check to determine if a given element is valid within the current element.

Object: "XML Data Object"

Return: Boolean - **True** if the element can exist under the current element.

Parameters:

strElement	String	The name of the element to check for its ability to exist under the current element.
------------	--------	--

Example:

```
if(true == objTag.ValidChildElment("ElemB"))
{
    alert("The ElemB element can exist here.");
}
```

Method: XMLProcessor

Description: Retrieves the interface to the XML Object. All advanced XML functionality is through this object. (Only available with eWebEditPro+XML.)

Object: "eWebEditPro ActiveX Control Object"

Master List of Properties

Property: CmdCaption

Description: Retrieves the caption. If a special button, the caption is a key word.

Object: "Command Item Object"

Property: CmdData

Description: If this is a list item, this property sets the current item to the entry that contains the long data value associated with the text. For a combo-box, it is either the long value given to the item or the index into the item. For a text box, it is the length of the string. For a toggle, it is the 1/0 (on/off) state.

Object: "Command Item Object"

Property: CmdGray

Description: If set to **true**, the command is disabled and displayed as a grayed image. The button does not produce a command when selected by the user. If set to **true**, the command is available to the user.

Object: "Command Item Object"

Property: CmdIndex

Description: This property only applies to list items. It sets the currently selected index and retrieves the currently selected index into the list box.

Object: "Command Item Object"

Property: CmdName

Description: This returns the command name associated with the button. If the command name of a list item is required, use `ListCommandName()`.

Object: "Command Item Object"

Property: CmdSorted

Description: Sets or retrieves whether the list box command referenced is a sorted list.

Object: "Command Item Object"

Property: CmdStyler

Description: Reflects the style of the command. The style is assigned when the command is created.

This is a read-only property.

Object: "Command Item Object"

Property: CmdText

Description: Sets the current selection for a list box. It sets the edit text for an edit box. The text is displayed on the button, no matter what.

Object: "Command Item Object"

Property: CmdToggledOn

Description: This property is only available to buttons that are created with the Toggle style. If the value is **true**, the button appears pressed in or selected. If false, it appears popped out or unselected.

Object: "Command Item Object"

Property: CmdToolTipText

Description: Contains the tooltiptext associated with a command. You can modify the tooltip through this property.

Object: "Command Item Object"

Property: CmdType As etbCommandStyles

Description: The command type assigned during the creation of the command.

This is a read-only property.

Object: "Command Item Object"

Property: CmdVisible

Description: Reflects the visibility of a command. If **true**, the user can see the command. If false, the command is invisible.

Do not use this property to disable buttons. Use the **CmdGray** property instead.

If the button is made invisible, an empty space replaces the button.

Object: "Command Item Object"

Property: MaxListboxWidth

Description: Sets or retrieves the width of an edit box or a list box in characters.

Object: "Command Item Object"

Property: ActiveSchema

Description: Selects or retrieves the namespace of the current schema.

If the value is set, the object searches for the given namespaces in all loaded schemas loaded. If the namespace is found, that schema becomes the active schema.

The value retrieved with this property is the namespace of the currently active schema.

If a schema is successfully selected, a cleaning operation automatically occurs to display the content according to the selected schema.

Schemas are loaded using the [AddSchema](#) method.

Object:"XML Object"

Type: String - the schema's namespace

See Also: "[Working with Schemas](#)" on page 666

Example: This example shows how to use the attribute to select the active schema.

```
function SelectSchema(sEditorName, sNameSpace)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();

    objXmlDoc.setProperty("ActiveSchema", sNameSpace);
    if(objXmlDoc.getPropertyInteger("ErrorCode") == 0)
    {
        alert("Schema successfully selected.");
    }
    else
    {
        alert("Error selecting schema: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

Property: AutoFillIn

Description: If this value is `true` and the user inserts an element contained within a loaded schema, all required elements within the inserted element are also inserted.

If the automatically inserted elements have required elements, they are also inserted. If the inserted elements and any required elements have required

attributes, these are included with the elements with either their default values, the first value in their value list, or as an empty value.

If this is false, only the selected element is inserted.

Object:"XML Object"

Type: Boolean

Property: ErrorCode

Description: Contains the error code for the last error. When an operation returns false, this value can be used to perform a check to see what the error is.

This long value is either an **eWebEditPro+XML** error value or an error value from the objects that **eWebEditPro+XML** uses, such as the MS XML parser.

Object:"XML Object"

Type: Long - the error value. A value of 0 means no error.

Example:

```
alert("Error: " + objXmlDoc.getPropertyInteger("ErrorCode"));
```

Property: ErrorFilePos

Description: When there is an error with content validation or loading a schema, this property will contain the position in the file where the error occurred.

Object:"XML Object"

Type: Long - The line where the error occurred.

Example:

```
alert("Error: " + objXmlDoc.getPropertyInteger("ErrorLine"));
```

Property: ErrorLine

Description: When there is an error with content validation or loading a schema, this property contains the line where the error occurred.

Object:"XML Object"

Type: Long - The line where the error occurred.

Example:

```
alert("Error: " + objXmlDoc.getPropertyInteger("ErrorLine"));
```

Property: ErrorPos

Description: If there is an error when validating content or loading a schema, this property contains the position in the line where the error occurred. The line containing the error is held in the ErrorLine property.

Object:"XML Object"

Type: Long - The position in the line where the error occurred.

Example:

```
alert("Error: " + objXmlDoc.getPropertyInteger("ErrorPos"));
```

Property: ErrorReason

Description: When an error occurs when validating content or loading a schema, this property contains a short reason, or description, why there was an error.

The description is in English, so if your users do not read English, use the other error properties to describe the error.

Object:"XML Object"

Type: String - A short description of the error given in English.

Example:

```
alert("Error: " + objXmlDoc.getPropertyString("ErrorReason"));
```

Property: ErrorSrcText

Description: If there is an error validating content or loading a schema, this property contains the source within the file that causes the error. This is not a description but the schema text that caused the error.

Object:"XML Object"

Type: String - The schema source code that has the error.

Example:

```
alert("Error: " + objXmlDoc.getPropertyString("ErrorSrcText"));
```

Property: ErrorURL

Description: If there is an error validating content or loading the schema, this value contains the URL of the loaded schema that contains the error.

Object:"XML Object"

Type: String - The URL of the schema with the error.

Example:

```
alert("Error: " + objXmlDoc.getPropertyString("ErrorURL"));
```

Property: RootTag

Description: Retrieves or sets the root tag of a loaded XML document.

Object: "XML Object"

Limitations:

- Only available when full XML documents are loaded
- The functionality is only useful when the root tag is hidden from the user
- The root tag must be in the root tag format (for example, <myroot attr1="hello">) and no ending tag (for example, </myroot>) is needed.
- The root tag name cannot be changed through this method. To change the root tag name, use the XmlHeader property.
- If the root tag name set here does not match the existing root tag name, an error occurs and all information being set (the attributes) is ignored.

Type: String

Example:

```
// This function will show what the current root tag values are.  
function ShowRootTag(sEditorName)  
{  
    var objXml = eWebEditPro.instances[sEditorName].editor.XMLProcessor();  
    window.document.frmeditor1.rootElemText.value = objXml.getPropertyString("RootTag");  
  
    // This function will take the modified root tag information and set it in the editor.  
    // Notice that an error is checked when the tag is set.  
    function LoadRootTag(sEditorName, sRootValue)  
    {  
        var objXml = eWebEditPro.instances[sEditorName].editor.XMLProcessor();  
        objXml.setProperty("RootTag", sRootValue);  
        if(objXml.getPropertyInteger("ErrorCode") != 0)  
        {  
            alert(objXml.getPropertyString("ErrorReason"));  
        }  
    }  
}
```

Property: ShowRootTag

Description: This setting is only for full XML documents. It determines whether or not the user can see the root tag and its existence in the header of the XML data.

If this value is **true**, the root tag is shown to the user for them to modify. In this case, it does not exist in the header information of the XML. The user may be able to edit the root tag directly, but he can also, unfortunately, add tags outside the root tag. This is because we support mixed content and must allow users to do this.

If this value is **false**, the root tag is hidden from the user. In this case, the root tag acts like the body tag when HTML is used. The root tag is held with the header information of the XML. Since the user cannot access the root tag, the user cannot insert tags outside of that tag. Unfortunately, the only way to edit the root tag is to have the client scripting use the getHeadHTML and setHeadHTML functions in the editor.

Object: "XML Object"

Type: Boolean - **True** if the root element is available to the user.

Example:

```
objXmlDoc.setProperty( "ShowRootTag" , false );
```

Property: TransformOnLoad

Description: Sets or reads the XSLT file that is used when whole content is loaded using the setDocument method in the editor. This is normally set in the configuration or xmlinfo data, but can also be done dynamically.

This must be a file and not a stream. You can identify the file by a relative path or a fully qualified path. It must be accessible to the instance of the editor.

This property is useful if a client script is selecting different content during an edit session. This allows you to specify a transformation to use when the content is loaded.

NOTE

[Ektron's eWebEditPro+XML+XML Knowledge Base](http://www.ektron.com/support/ewebeditprokb.cfm) contains several articles that describe using XSLTs. To read the articles, go to <http://www.ektron.com/support/ewebeditprokb.cfm> and enter XSLT into the Search field.

Object: "XML Object"

Type: String - the XSLT file's location.

Example:

```
objXmlDoc.setProperty( "TransformOnLoad" , "myxform.xsl" );
```

Property: TransformOnSave

Description: Sets or reads the XSLT file that is used when whole content is extracted from the editor using the getDocument method. This is normally set in the configuration or xmlinfo data, but can also be done dynamically.

This must be a file and not a stream. You can identify the file by a relative path or a fully qualified path. It must be accessible to the instance of the editor.

This property is useful if the client script is selecting different content during an edit session. This allows it to specify a transformation to use when the content is extracted for saving or display.

Object: "XML Object"

Type: String - The location of the XSL file.

Example:

```
objXmlDoc.setProperty( "TransformOnSave" , "myxform.xsl" );
```

Property: XmlHeader

Description: This property is only valid with full XML documents. This offers an alternative, and more direct, method of accessing the XML header information.

The other method is using the getHeadHtml and setHeadHtml methods in the editor. This property allows the setting and retrieving of the header for a loaded, full XML document.

This does not work with loaded HTML files.

IMPORTANT NOTE

The powerful `showroot` attribute in the `tagdefinitions` element (which is also available in the `ShowRootTag` property) affects what is returned by this attribute. When retrieving the header, these rules apply:

- If the root element is shown, when the header is retrieved, it will not contain the root element.
- If the root element is not shown, the root element can be retrieved, modified, and set through the header retrieval methods.

When setting the header information, the 'showroot' attribute is not as important for this property, but it still has some effect. These rules apply:

- If the root element is given, it will be used unless the root is shown to the user.
- If the root element is not given when this method is used, the existing root information is preserved and used when the header and full content are retrieved.

Object: "XML Object"

Type: String - The header information for a loaded XML document.

Example: Here is an example of how this property can be used.

```
alert(objXmlDoc.getPropertyString( "XmlHeader" ));
```

Here are some example XML headers that can be returned by this call.

```
<?xml version="1.0"?>
```

```
<?xml version="1.0"?>
<x:catalog xmlns:x="urn:books" xmlns="urn:books"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:books booksv4.xsd">
```

```
<?xml version="1.0"?>
<x:catalog>
```

```
<?xml version="1.0"?>
<!DOCTYPE x:catalog SYSTEM "books.dtd">
```

```
<?xml version="1.0"?>
<!DOCTYPE x:catalog SYSTEM "books.dtd">
<x:catalog>
```

Property: AllowEdit

Description: Determines if the user can edit the tag contents. A read-only property.

Object: "XML Data Object"

Type: Boolean - **True** if the contents can be edited.

Example:

```
function ShowTagEditable(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    alert("Editable = " + objTag.getPropertyBoolean("AllowEdit"));
}
```

Property: Attributes

Description: The attributes of a tag can be affected using this property. When retrieved, they are formatted as they appear within the tag. The delimiters are white spaces between the attribute/value pairs and equal signs (=) between the attribute name and the value.

When attributes are assigned, they must also follow this convention.

This does not return the list of valid attributes. This returns the attributes currently used by the specific tag in the XML data. To get the list of valid attributes, use the ElementAttributes method.

See Also: "Retrieving and Setting Attributes" on page 673

Object: "XML Data Object"

Type: String - The attributes as seen in the tag.

Example:

```
function ShowTagEditable(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    objTag.setProperty("Attributes", "attr1='hello' attr2='world'");
}
```

Property: Content

Description: Retrieves or sets the content within the tag. This includes any text or tags within this tag.

Object: "XML Data Object"

Type: String - The content of the tag.

Example:

```
alert(objTag.getPropertyString("Content"));
```

Property: DataStyle

Description: Read-only

This property contains the style parameters used to display the content of the tag. Any style parameters not specified here are inherited from the style parameters contained in the TagStyle attribute.

Style values are defined in the configuration data or in the xmlinfo data for each tag.

See Also: "Where to Define the Appearance of XML Tags"; "Property: TagStyle"

Object: "XML Data Object"

Type: String - The style values.

Example:

```
alert("Style: " + objTag.getPropertyString("DataStyle"));
```

Here are some sample style values returned.

```
background-color:#ccccff; text-decoration:underline;
font-family:arial; font-weight:normal; background-color:#cccccc;
border:solid blue 1pt; margin:2px; width:95%;" dstyle="font-
```

```
family:arial; font-weight:normal; background-color:white;  
padding:2px; width:95%;
```

Property: Description

Description: Read-only

Returns the description of the tag displayed to the user.

Object: "XML Data Object"

Type: String - The description of the tag shown to the user.

Example:

```
alert(objTag.getPropertyString("Description"));
```

Property: Icon

Description: Read-only

This property contains the path of the icon shown with the tag description. If the icon is not shown, this value is empty.

Object: "XML Data Object"

Type: String - The icon shown with the tag description.

Example:

```
alert("The icon is " + objTag.getPropertyString("Icon"));
```

Property: ShowIcon

Description: Read-only

This property contains whether an icon is shown to the user with the tag. The Icon property contains the path to the image shown. If the Icon property does not contain a value, an icon is not shown.

Object: "XML Data Object"

Type: Boolean - **True** if the icon is shown.

Example:

```
if(objTag.getPropertyBoolean("ShowIcon") == true)  
{  
    alert("An icon is shown with the element.");  
}
```

Property: ShowName

Description: Read-only

Contains whether the name, or element description, is shown with the element. A name is not required to show an icon with an element and vice versa.

Object: "XML Data Object"

Type: Boolean - **True** if the description is shown.

Example:

```
if(objTag.getPropertyBoolean("ShowName") == true)
{
    alert("The description is shown with the element.");
}
```

Property: TagName

Description: Read-only

Contains the name of the element. This is the actual name used within the XML data content. The case of the element name is preserved.

Object: "XML Data Object"

Type: String - The name of the element.

Example:

```
alert("Using element " +
objTag.getPropertyString("TagName"));
```

Property: TagStyle

Description: Read-only

Contains the style parameters used to display the tag. These style values pertain to the description area, the border, and default style values used when displaying attributes and content to the user.

Style values are defined in the configuration data or in the xmlinfo data for each tag.

Object: "XML Data Object"

Type: String - The style values.

Example:

```
alert("Style: " + objTag.getPropertyString("TagStyle"));
```

Here are some sample style values returned:

```
background-color:#ccccff; text-decoration:underline;
font-family:arial; font-weight:normal; background-color:#cccccc;
border:solid blue 1pt; margin:2px; width:95%;" dstyle="font-
family:arial; font-weight:normal; background-color:white;
padding:2px; width:95%;
```

Property: TagType

Description: Read-only

Contains the type of flag that defines how the element is arranged when formatted. The value can be one of these:

- 0 - Empty
- 1 - Nonempty
- 2 - Vertical
- 3 - Horizontal

See Also: ["Types of XML Tags"](#)

Object: ["XML Data Object"](#)

Type: Long - The value pertaining to the type of tag arrangement.

Example:

```
alert("Type: " + objTag.getPropertyInteger("TagType"));
```

Property: TransferMethod

Description: This property specifies how the Automatic Upload mechanism performs an upload when local files are detected. This value can be any of the values defined in the `type` attribute of the `autoupload` element. (*See Also:* ["Type"](#))

If a server is specified in a receiving page's path, that server is used.

```
objAuto.setProperty("TransferMethod", strAutoURL);
```

Object: ["XML Data Object"](#)

Property: ServerName

Description: This property specifies the server to use with the receiving page. It is not needed if the server is specified with the receiving page in the `TransferMethod` property.

```
objAuto.setProperty("ServerName", strRcvServer);
```

Object: ["Automatic Upload Object"](#)

Property: LoginName

Description: The login name of the user uploading the image. This may be encrypted in the configuration data.

Object: ["Automatic Upload Object"](#)

Type: String

Property: LoginRequired

Description: Enables or disables the act of logging into a remote site. This property must be set to **True** to activate the login name and password transmission to the server.

Object: "Automatic Upload Object"

Type: Boolean

Property: Password

Description: The password of the user uploading the image. This may be encrypted in the configuration data.

Object: "Automatic Upload Object"

Type: String

Property: TransferRoot

Description: The same as "Property: DefDestinationDir" on page 150.

Object: "Automatic Upload Object"

Type: String

Property: ValidExtensions

Description: The file extensions of images that can be uploaded, entered as a comma-delimited string. For example

"gif,tif,jpg"

Object: "Automatic Upload Object"

Type: String

Property: WebRoot

Description: The base location for accessing uploaded images from a Web page.

For example `http://www.ektron.com/images`

Object: "Automatic Upload Object"

Type: String

Property: ContentDescription

Description: Contains the description string that is sent to the server when the content is posted.

Object: "Automatic Upload Object"

Type: String

Example

```
var objAutoUpload =
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
objAutoUpload.setProperty("ContentDescription", "News Article");
```

Property: AllowUpload

Description: Enables or disables automatic upload feature.

Object: "Automatic Upload Object"

Type: boolean

Property: ContentTitle

Description: The title of the content posted to the server. This value is posted with the content.

Object: "Automatic Upload Object"

Type: String

Example

```
var objAutoUpload =
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();
objAutoUpload.setProperty("ContentTitle", "Man Bites Dog");
```

Property: ContentType

Description: The type of the content posted to the server. Valid types for this property are the same as the GetContent and SetContent methods.

See Also: "Content Type Categories"

Object: "Automatic Upload Object"

Type: String

Example

```
var objAutoUpload =  
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();  
objAutoUpload.setProperty("ContentType", "htmlbody");
```

Property: Port

Description: The port used for the HTTP posting or the FTP transfer. If the value is 0, the default port for the upload process is used.

Object: "Automatic Upload Object"

Type: Long

Example

```
var objAutoUpload =  
    eWebEditPro.instances[g_sEditorName].editor.MediaFile().AutomaticUpload();  
objAutoUpload.setProperty("port", 80);
```

Property: Alignment

Description: The image's alignment on the page. Possible values are:

- left
- right
- top
- middle
- bottom
- AbsMiddle
- AbsBottom

For documentation of the alignment values, please refer to the "Inserting Images" chapter in the **eWebEditPro+XML** User Guide.

The user can edit this value in the Picture Properties dialog box.

Object: "Parameters Object"

Type: String

Property: AllowSubDirectories

Description: Determines whether or not a user can select sub-directories. If false, the user cannot.

Currently set but not implemented.

Object: "Parameters Object"

Type: Boolean

Property: allowupload

Description: If **true**, the user can upload files from the local PC to the server. If **false**, the user can only insert files that reside on the server.

NOTE

It is up to the upload mechanism to use this value. For FTP, if this value is **false**, FTP does not let the user upload files. It only lists the available files. The ASP and ColdFusion samples work the same way. If the value is **false**, the upload frame is blank.

Type: Boolean

Object: "Parameters Object"

Example: `objEditor.setProperty("AllowUpload", true);`

Property: BaseURL

Description: The base URL value set in the editor. This is a friend property. It should be set by a routine that knows the base URL.

Type: String

Object: "Parameters Object"

Property: BorderSize

Description: The size of the image's border in pixels. The user can edit this value in the Picture Properties dialog box.

Type: Integer

Object: "Parameters Object"

Property: DefDestinationDir

Description: The destination path to where the image will be placed. This is the same as the [TransferRoot](#).

Type: String

Object: "Parameters Object"

Property: DefSourceDir

Description: The initial directory that appears when the user is selecting a local file.

Type: String

Object: "Parameters Object"

Property: Domain

Description: The domain name of the upload server. This is mainly for use with FTP, but may also be important for other upload mechanisms.

Type: String

Object: "Parameters Object"

Property: FileSize

Description: The size of the image file in bytes. This value is set when the user selects a local file.

Type: Long

Object: "Parameters Object"

Property: FileTitle

Description: The title of the file. This is not the file name but a descriptive title that users assign after selecting the file. It is used as the image's alt text.

Type: String

Object: "Parameters Object"

Property: FileType

Description: The type of file. These are the choices.

- bitmap
- video
- audio
- document
- other

Type: String

Object: "Parameters Object"

Property: FWLoginName

Description: User's login name for the firewall. Not currently used.

Type: String

Object: "Parameters Object"

Property: FWPassword

Description: User's password for the firewall. Not currently used.

Type: String

Object: "Parameters Object"

Property: FWPort

Description: The firewall port to use for any transfer. If this value is zero (0), the transfer mechanism determines the port. Not currently used.

Type: Integer

Object: "Parameters Object"

Property: FWProxyServer

Description: Firewall proxy server. Not currently used.

Type: String

Object: "Parameters Object"

Property: FWUse

Description: If true, a firewall mechanism is used. Not currently used.

Type: Boolean

Object: "Parameters Object"

Property: FWUsePassV

Description: If true, PASV mode FTP is enabled.

Type: Boolean

Object: "Parameters Object"

Property: Get ShowResolutionOverride

Description: If set to **true** or returns **true**, the user is offered the check box to manually enable or disable the path resolution mechanism.

Type: Boolean

Object: "Parameters Object"

Property: Get EnablePathResolution

Description: If set to **true** or returns **true**, the path resolution functionality is enabled. If disabled, it is the responsibility of the user or administrator to properly set the path.

Type: Boolean

Object: "Parameters Object"

Property: Get XferType

Description: Retrieves or sets the transfer type string. A developer can use this to dynamically change the transfer type. For example, the developer can go from FTP to loading a Web page through this value.

Type: String

Object: "Parameters Object"

Property: Get IsValid

Description: Returns whether the current upload connection is valid. If there are problems connecting to the upload location or the connection has not been tried, this is "**false**".

Type: Boolean

Object: "Parameters Object"

Property: HandledInternally

Description: The upload has already been handled internally. If **true**, the upload is skipped, and only the notification is done.

Type: Boolean

Object: "Parameters Object"

Property: HorizontalSpacing

Description: Horizontal spacing attribute to use in the HTML. The user can edit this value in the Picture Properties dialog box.

Type: Integer

Object: "Parameters Object"

Property: ImageHeight

Description: The height of the image. This value is set when an image is selected.

See Also: [ShowHeight](#)

Type: Integer

Object: "Parameters Object"

Property: ImageWidth

Description: The width of the image. This value is set when an image is selected. This is not a rendered size, but the actual size of the image.

See Also: [ShowWidth](#)

Type: Integer

Object: "Parameters Object"

Property: IsLocal

Description: Set this to **true** if a local file will be placed into the [SrcFileName](#) property.

The object processes the path information differently for local files.

If this value is not set, the object resolves the source location to a remote path, and upload is not possible.

Type: Boolean

Object: "Parameters Object"

Property: LoginName

Description: The login name of the user uploading the image. This may be encrypted in the configuration data.

Type: String

Object: "Parameters Object"

Property: MediaType

Description: This property determines which file extensions are available in the Media File Selection dialog. It overrides the list of extensions provided in the configuration data.

This property has three possible values: images, nonimages, or all.

- images allows only the following extensions: gif,jpg,png,jpeg,jif,bmp,tif,tiff
- nonimages allows any extension other than images
- all allows all file extensions

Type: String

Object: "Media File Object"

Property: MaxFileSizeK

Description: The maximum size in kilobytes of an image to be uploaded. A value of zero (0) means no size limit.

Type: Integer

Object: "Parameters Object"

Property: NeedConnection

Description: A read-only property that determines if a connection is necessary with the current upload method.

Type: Boolean

Object: "Parameters Object"

Property: Password

Description: The password of the user uploading the image. This may be encrypted in the configuration data.

Type: String

Object: "Parameters Object"

Property: Port

Description: The port to use for uploads. If zero (0), the file's upload type determines the port.

Type: Integer

Object: "Parameters Object"

Property: ProxyServer

Description: The name of the proxy server to use with uploads. This property is not required.

Proxy servers are primarily used with FTP.

Type: String

Object: "Parameters Object"

Property: RemotePathFileName

Description: The remote path and name of the currently selected file. This path may have been generated using the path parameters when a local file is entered into [SrcFileLocationName](#).

The application can also set a remote path and name to override the generated one.

Type: String

Object: "Parameters Object"

Property: ResolveMethod

Description: The method by which the image source path is resolved. The choices are:

- FULL - fully qualified to server
- HOST - relative to host
- LOCAL - relative to page
- GIVEN - relative to given location - WebRoot

See Also: "[Using Local or Given Image Path Resolutions](#)" on page 486

Type: String

Object: "Parameters Object"

Property: ResolvePath

Description: The path used to resolve an image path when GIVEN is the resolution method. It defaults to the WebRoot, since files are uploaded there.

Type: String

Object: "Parameters Object"

Property: ShowHeight

Description: The height attribute of the HTML image tag.

Enter a value here if you want to determine the image's height, regardless of its actual size (which is stored in the [ImageHeight](#) property).

This value defaults to the ImageHeight property value.

The user can edit this value in the Picture Properties dialog box.

Type: Integer

Object: "Parameters Object"

Property: ShowWidth

Description: The width attribute for the HTML image tag.

Enter a value here if you want to determine the image's width, regardless of its actual size (which is stored in the [ImageWidth](#) property). This value defaults to the ImageWidth property value.

The user can edit this value in the Picture Properties dialog box.

Type: Integer

Object: "Parameters Object"

Property: SrcFileLocationName

Description: The full location of the source file. This includes the server, if applicable, and the path and file name with extension.

Type: String

Object: "Parameters Object"

Property: TransferMethod

Description: The name of the upload method used if the `ProvideMediaFile` method is called. The value of this parameter determines what the upload mechanism should do.

The string can be anything from a key word to a URL. If it is not an internal value, a script must interpret it. The internal values are `FTP` and `FILE`.

For more information on `FILE`, see "[Setting up an Image Repository](#)" on [page 510](#).

Type: String

Object: "Parameters Object"

Example:

```
function ReadTransferMethod()
{
    var objAutoUpload = GetAutoUploadObject();
    return objAutoUpload.getPropertyString("TransferMethod");
```

Property: TransferRoot

Description: The same as the `DefDestinationDir`.

Type: String

Object: "Parameters Object"

Property: UsePassV

Description: If `true`, FTP works in passive mode.

Type: Boolean

Object: "Parameters Object"

Property: ValidConnection

Description: If `true`, the system has made a valid connection with the current connection parameters.

Type: Boolean

Object: "Parameters Object"

Property: ValidExtensions

Description: The file extensions of images that can be uploaded, entered as a comma-delimited string. For example

"gif,tif,jpg"

Type: String

Object: "Parameters Object"

Property: VerticalSpacing

Description: The value of the vertical spacing attribute of the HTML image tag.

The user can edit this value in the Picture Properties dialog box.

Type: Integer

Object: "Parameters Object"

Property: WebPathName

Description: The Web accessible name of the specified file. The name is resolved using the rules assigned to the ResolveMethod value specified.

For example `http://www.ektron.com/images/me.gif`.

Type: String

Object: "Parameters Object"

Property: WebRoot

Description: The base location for accessing uploaded images from a Web page.

For example `http://www.ektron.com/images`.

Type: String

Object: "Parameters Object"

Property: bodyStyle

Object: "eWebEditPro ActiveX Control Object"

Description: Specifies Cascading style sheet (CSS) attribute values, such as background color, default font style, size, color and more. The bodyStyle parameter sets any valid CSS style supported by your browser.

Note that this property affects the same values as the style attribute in the body tag, for example:

```
<body style="background-color: black; font-size: 24pt">
```

If both are set, the bodyStyle parameter takes precedence.

Effect of Style Sheet on bodyStyle Parameter

If a style sheet is being referenced by the editor, the style sheet's specifications override any font styles defined in the bodyStyle parameter except for the BODY element. So, for example, while you can set the default font using bodyStyle, that setting will have no effect on text within the <P> tag.

If a style sheet is being used, you have three options for resolving the clash of style specifications between the bodyStyle parameter and the style sheet.

- disable the style sheet
- change the style sheet so that it specifies the desired default font
- change the style sheet so that it does not specify font attributes for the content. The style sheet can continue to specify other attributes, such as page break after, widow/orphan control, and margins

When to Set the Parameter

This parameter may be set before or after the editor is created. It may also be changed while the editor is loading, and the change will apply immediately.

Examples

Examples of how to set the body style property appear below.

In ewebeditprodefaults.js

```
this.bodyStyle = "background-color: black; font-size: 24pt";
```

JavaScript Parameter Before the Editor is Created

```
eWebEditPro.parameters.bodyStyle = "background-color: black; font-size: 24pt";
```

Property: CharSet

Description: Specifies the charset value for a page. For example

```
<meta content="text/html; charset=iso-8859-1">.
```

This is only needed if saving the entire document.

See Also: ["Encoding Special Characters" on page 417](#) and ["Implementing a Web Site that Uses UTF-8 Encoding" on page 427](#)

Object: ["eWebEditPro ActiveX Control Object"](#)

Property: Config

Description: Specifies the URL of the config XML file. Although this ActiveX control property can contain the XML content, it typically refers to an XML file. (For details, see ["Managing the Configuration Data" on page 251](#).)

Object: ["eWebEditPro ActiveX Control Object"](#)

Property: Disabled

Description: Specifies when set to `true`, the editor is disabled, that is, content cannot be edited and all toolbar items are inactive.

You can use this property instead of `ReadOnly` if you want to make the whole editor inaccessible to the user. (With the `ReadOnly` property, the entire toolbar is not disabled.)

See Also: ["Property: ReadOnly" on page 162](#)

This is a boolean type field with a default value of `False`.

Object: ["eWebEditPro ActiveX Control Object"](#)

Examples

There are two ways to get access to the editor property at load time:

- When the editor is created, if you know what the property setting should be, set it in the parameters object before creating the editor.

See Also: ["The Parameters Object" on page 306](#)

In the examples below, the disabled property starts with a lower case letter. Properties and methods that are *not* directly accessed in the editor [object.editor...] always start with a lower case letter. So, to access the disabled property through the JavaScript parameters object, begin with a lower case letter:

```
[eWebEditPro.parameters.disabled = true]
```

On the other hand, to access the property through the editor, begin with an upper case letter:

```
[eWebEditPro.instances["myeditor"].editor.Disabled = true]
```

or

```
object.editor.setProperty("Disabled", true)]
```

Here is a complete example.

```
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{
    eWebEditPro.parameters.disabled = true;
    eWebEditPro.create("MyContent1", "100%", 400);
}
//-->
</script>
```

- If you need an external function to set the editor parameters, use the oncreate event, which is called just before the editor is created.

```
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{
    eWebEditPro.addHandler("oncreate", "initEditorValues('MyContent1')");
    eWebEditPro.create("MyContent1", "100%", 400);
}
function initEditorValues(sEditorName)
{
    eWebEditPro.parameters.disabled = true;
}
//-->
</script>
```

If at run time, after the editor is created and operational, you want to disable the editor, set the disabled property to true. To later enable it, change it to false.

```
<script language="JavaScript1.2">
<!--
function DisableEditor(sEditorName, bDisabled)
{
    eWebEditPro.instances[sEditorName].editor.setProperty("disabled", bDisabled);
}
//-->
</script>
```

Property: Get WDDX

Description: Sets or retrieves assigned WDDX data. This is to maintain version 1.8 compatibility.

Object: "eWebEditPro ActiveX Control Object"

Property: hideAboutButton

Description: Set to **True** to remove the About () button from the toolbar.

Object: "eWebEditPro ActiveX Control Object"

Property: IsDirty

Description: This property returns **true** if the content has changed, **false** if no changes were made to content in this editor.

For more information, see "[Method: isChanged](#)" on page 95.

Object: "eWebEditPro ActiveX Control Object"

Property: License

Description: The license keys of the editor. Separate each with a comma.

Ektron provides these keys after purchase. For development purposes, the license keys for 127.0.0.1 and localhost are built into the editor.

NOTE

eWebEditPro+XML displays an "Invalid License" message if the license key is improperly entered.

Object: "eWebEditPro ActiveX Control Object"

Property: Locale

Description: The URL of the localization directory or file. (For details, see "[Modifying the Language of eWebEditPro+XML](#)" on page 265.)

Object: "eWebEditPro ActiveX Control Object"

Property: ReadOnly

Description: see "[Property: readOnly](#)" on page 170

Object: "eWebEditPro ActiveX Control Object"

Property: SrcPath

Description: Where eWebEditPro+XML is installed, that is, the eWebEditProPath. The configuration data has an environment variable [eWebEditProPath], which is replaced by the value in srcPath.

Do not change the value of srcPath.

Object: "eWebEditPro ActiveX Control Object"

Property: StyleSheet

Description: Which style sheet file (CSS) to apply to the editor content. If the entire document is retrieved from the editor, the style sheet is included in the head section using the link element.

The value of the StyleSheet property is the value of the link tag's href attribute.

Object: "eWebEditPro ActiveX Control Object"

Examples

- via parameter before the editor is created:

```
eWebEditPro.parameters.styleSheet = "mystyle.css";
```

- via ActiveX after the editor is loaded and ready:

```
eWebEditPro.myEditor1.setProperty("StyleSheet", "mystyle.css");
```

Both examples produce this line between the document's head tags:

```
<link rel="stylesheet" type="text/css" href="mystyle.css">
```

If only the content within the body tags is retrieved, you need to also apply the style sheet to the HTML document that displays the content.

If you change the StyleSheet property, the changes are visible immediately.

See Also: ["Style Sheets" on page 430](#)

Property: Title

Description: A document title for the page. For example

```
<head>
<title>Ektron, Inc. - dynamic Web content management - html editor</title>
```

This is only needed if saving the entire document.

Object: "eWebEditPro ActiveX Control Object"

Property: versionInstalled

Description: Retrieves the version of the control. It is a comma delimited list with this format:

Major Major, Minor Major, Major Minor, Minor Minor
Or
Version, Point Release, 0, Revision
(The Major Minor value is not used, so it is always 0.)

Object: "eWebEditPro Object"

Syntax

```
strData = [form!]ewebeditpro5.Version
```

Remarks

The Major Minor value of 0 is in the format because of the agreed upon format for software object versions. If comparing versions, the string must be parsed and each item converted to an integer.

Examples

Displays the control version:

```
function ShowVersion()  
{  
    alert(testItem1.Version);  
}
```

or

```
alert(eWebEditPro.instances.MyContent1.editor.version);
```

or

```
alert(eWebEditPro.instances.MyContent1.editor.getPropertyString("version"));
```

or

```
var strVersion = "unknown";  
if (eWebEditPro.versionInstalled)  
{  
    strVersion = eWebEditPro.versionInstalled;  
}  
document.write("Version of ewebeditpro5.ocx actually installed: <span class=value>" +  
strVersion + "</span><br>");
```

Currently only available with IE.

Property: xmlInfo

NOTE This method is used with **eWebEditPro+XML** only.

Description: Dynamically assigns XML and custom tag configuration data that is external to the normal configuration data.

See Also: "The **xmlInfo** Property" on page 649

Object: "eWebEditPro ActiveX Control Object"

Property: border

Description: The border attribute of the popup edit button.

Object: "Image Tag Object"

Type: integer

Property: height

Description: The height attribute of the popup edit button.

Object: "Image Tag Object"

Type: integer

Property: width

Description: The width attribute of the popup edit button.

Object: "Image Tag Object"

Type: integer

Property: src

Description: The source attribute of the popup edit button.

Object: "Image Tag Object"

Property: alt

Description: The source of the image that appears on the popup edit button.

Object: "Image Tag Object"

Property: Start

Description: Determines the beginning of the HTML that appears on the popup edit button.

Object: "Button Tag Object"

Property: End

Description: Determines the end of the HTML that appears on the popup edit button.

Object: "Button Tag Object"

Property: Type

Description: Determines the form of the popup edit button.

Object: "Button Tag Object"

Property: tagAttributes

Description: Use to assign custom attributes to the popup edit button.

Object: "Button Tag Object"

Property: value

Description: Determines the value of the popup edit button.

Object: "Button Tag Object"

Property: baseURL

Description: This property sets the current URL of the editor to the specified location. All references are based on, and relative to, this location. The control uses this value to find items to display.

This property does *not* need to be set. If it is not set, the ActiveX control determines its current location. The property is useful for allowing a script to point the editor at another location.

IMPORTANT!

A trailing slash is required.

Also, if image paths are relative, you must set the `xferdir` and `webroot` attributes of the `mediafiles` element to match this value. Otherwise, the transfer and reference directories may not be on the same server, and the current URL defaults to a full path resolution.

Type: String

Object: "eWebEditPro ActiveX Control Object"

Example

Change to match the setting of the baseURL in the config.xml data.

```
<features>
    .
    .
<mediafiles>
    .
    .
    <transport>
        .
        .
        <xferdir src="http://msimg.com/w"/>
        <!-- set to new location -->
        <webroot src="" />
        <!-- Keep blank so it matches xferdir -->
```

```
        </transport>
    </mediafiles>
</features>
```

Example addition in the script that can change this value.

```
function myOnReady(sEditorName)
{
    eWebEditPro.instances[sEditorName].setProperty("BaseUrl", "http://msimg.com/w");
}
```

Property: type

Description: The name of the current event without the “on” prefix. The type is always lowercase.

Object: “Event Object”

Example:

```
if (eWebEditPro.event.type == "dblclickelement")
{
    ...
}
```

Property: srcName

Description: The name of the instance of the editor that is the source of the current event.

Object: “Event Object”

Example:

```
onDblClickElementHandler(oElement)
{
    // Replace element that was double-clicked with a horz line.
    eWebEditPro.instances[eWebEditPro.event.srcName].editor.pasteHTML("<hr>");
    or
    eWebeditPro[eWebeditPro.event.srcName].pasteHTML("<hr>");
```

Property: buttonTag

Description: Object consisting of

- eWebEditProDefaults.buttonTagStart
- eWebEditProDefaults.buttonTagEnd
- eWebEditProMessages.popupButtonCaption

If null, the popup button does not appear.

See Also: “[Button Tag Object](#)”

Object: "Parameters Object"

Property: clientInstall

Description: The directory in which the client installation file (ewebeditproclient.exe) resides.

Object: "Parameters Object"

Property: cols

Description: The number of columns in the TEXTAREA element if eWebEditPro+XML is not installed or not supported. If undefined, the number of columns approximates the width specified when the browser is created.

Object: "Parameters Object"

Property: embedAttributes

Description: Optional attributes to the EMBED tag. Applies only to Netscape.

Object: "Parameters Object"

Property: maxContentSize

Description: The largest number of characters that can be saved in the editor window. If a user enters content that exceeds this size, an error message appears.

The maximum size of the content may be increased in some circumstances. Several factors affect the maximum size allowed.

- Netscape 4 fields are limited to 64K, that is 65535.
- ColdFusion limits the results received from ODBC queries' columns to 64K for performance reasons. It may be possible to edit ColdFusion's settings of your ODBC data source. Refer to your ColdFusion documentation for more information.
- The Web server may limit the size of form variables (for example, hidden fields), although typically the size is very large.
- If using a database, the database field type may be limited in size (for example, 64K bytes). Check your database documentation.
- If using ODBC, the ODBC driver on the server may limit the content.

Also, you may want to limit content size as a matter of corporate policy, personal preference, or to implement quotas where a user has a limited amount of space allocated.

If none of these restrictions applies to your situation (for example, all users have Internet Explorer), you can increase the value of maxContentSize in ewebeditprodefaults.js or set it in JavaScript.

To have no limit, set maxContentSize = 0.

NOTE

This parameter checks the number of characters, which may be different from the number of bytes, depending on the encoding method.

Object: "Parameters Object"

Property: objectAttributes

Description: Optional attributes to the OBJECT tag. Applies only to Internet Explorer. For example, the OBJECT tag has an attribute 'standby' that the developer could set to a string.

```
objectAttributes="standby='Please wait while the editor initializes.'";
```

Object: "Parameters Object"

Property: path

Description: The path to the **eWebEditPro+XML** files relative to the hostname. For example, /ewebeditpro5/.

This value is set in the ewebeditpro.js file.

Object: "Parameters Object"

Property: preferredType

Description: Specifies the type of editor to create. This property has three possible values:

- textarea - creates a standard HTML textarea field
- section - creates an edit button which, when pressed, displays a popup window with **eWebEditPro+XML**
- activex - creates the **eWebEditPro+XML** editor

If the editor was not installed on the client and the value is set to **section** or **activex**, an Edit button appears on the page. When the user clicks the button, he is prompted to download **eWebEditPro+XML**.

Examples:

```
In ewebeditprodefaults.js: this.preferredType = "textarea";
```

Or, on the page with the editor before it is created:

```
eWebEditPro.parameters.preferredType = "textarea";
```

Object: "Parameters Object"

Property: readOnly

Description: Prevents the user from modifying the editor content. This property is useful when you want the editor to display content that a user should not change.

You can set the parameter before creating the editor in JavaScript, or at run-time using the JavaScript **Instance object** or ActiveX property.

```
eWebEditPro.parameters.readOnly = true;  
eWebEditPro.instances[sEditorName].setReadOnly(true);  
eWebEditPro.instances[sEditorName].getReadOnly();  
eWebEditPro.instances[sEditorName].editor.setProperty("ReadOnly", true);
```

If you set **ReadOnly** to **true** from the client script, the editor content becomes read-only, and all toolbar buttons become inactive and ignore any API call or user selection.

Set the **ReadOnly** property to **false** to enable editing of the content and the toolbar buttons.

The **ReadOnly** property is *not* available if in Data Design or Data Entry mode. See Also: "[Supporting the Data Designer](#)" on page 601

The **readOnly** parameter and JavaScript Instance object methods, **setReadOnly** and **getReadOnly**, are compatible with the TEXTAREA field that displays if **eWebEditPro+XML** is not supported.

Object: "Parameters Object"

Property: rows

Description: The number of rows in the TEXTAREA element if **eWebEditPro+XML** is not installed or not supported.

If undefined, the number of rows approximates the height specified when the editor is created.

Object: "Parameters Object"

Property: textareaAttributes

Description: Optional attributes to the TEXTAREA tag. Apply only when a textarea field appears in place of **eWebEditPro+XML**, typically because the operating system does not support **eWebEditPro+XML**.

You can specify the row and column attributes of the textarea field using the rows and cols parameters. For example, you could use the **textareaAttributes** property to specify an onchange attribute value. For example

```
textareaAttributes = "onchange='mychangehandler()'" ;
```

Object: "Parameters Object"

Property: popup

Description: Lets you pass four parameters to the popup Web page (specified in `popupUrl` property).

- `url` (see “[Property: url](#)” on page 173)
- `windowName` (see “[Property: windowName](#)” on page 174)
- `windowFeatures` (see “[Property: windowFeatures](#)” on page 173)
- `query` (see “[Property: query](#)” on page 173)

By default, the `popupUrl` page is a static HTML page, but it could be a dynamically generated page. In either case, you may want to pass additional information to the popup page. For example, you may want to display the number of times the content has been edited, the title of the content, or anything else.

Here is an example that passes a title and instructions relevant to the content being edited.

On the page with the popup button:

```
<script language="JavaScript">
var sTitle = "Summary Description";
var sInstr = "Please enter a paragraph summarizing the page.";
with (eWebEditPro.parameters.popup)
{
    url = "cif_t0007popup.htm";
    windowName = "";
    windowFeatures = "location,scrollbars,resizable";
    query = "title=" + escape(sTitle) + "&instr=" + escape(sInstr);
    ...
    eWebEditPro.createButton(...);
}</script>
```

NOTE

The JavaScript `escape()` function ensures the text is saved to pass in a URL. For example, it changes all space characters to %20. The `unescape()` function restores the text.

On the popup page:

```

<script language="JavaScript">
var objQuery = new Object();
var strQuery = location.search.substring(1);
var aryQuery = strQuery.split("&");
var pair = [];
for (var i = 0; i < aryQuery.length; i++)
{
    pair = aryQuery[i].split("=");
    if (pair.length == 2)
    {
        objQuery[unescape(pair[0])] = unescape(pair[1]);
    }
}
</script>

```

Object: "Parameters Object"

Property: url

Description: Specifies the URL of a Web page to display in a popup window when an automatic installation is expected.

Example in ewebeditprodefault.js

```
this.installPopupUrl = this.path + "clientinstall/intro.htm";
```

See Also:

- ["Client Installation Pages" on page 297](#)

Object: "InstallPopup Object"

Property: windowName

Description: Specifies the name of the popup window. Typically, this is left as an empty string.

Example in ewebeditprodefault.js:

```
this.installPopupWindowName = " ";
```

Object: "InstallPopup Object"

Property: windowFeatures

Description: Specifies the popup window features as defined for the standard JavaScript window.open() method. (For more details on the JavaScript window.open() method, see a JavaScript reference.)

Example in ewebeditprodefault.js

```
this.installPopupWindowFeatures = "height=540,width=680,resizable,scrollbars,status";
```

See Also: "Property: windowFeatures" on page 173

Object: "InstallPopup Object"

Property: query

Description: An optional parameter that specifies query string values to pass to the page specified by the URL parameter. Typically, the query property is left as an empty string.

If specified, the query string is appended to the URL, separated by a question mark (?) character. Do not include the ? in the query string value.

Example in ewebeditprodefault.js

```
this.installPopupQuery = "";
```

Example in JavaScript

```
eWebEditPro.parameters.installPopup.query = "firstname=Bob&lastname=Smith";
```

Object: "InstallPopup Object"

Property: url

Description: The URL to the Web page that contains the editor that appears in the popup window. The default value is

```
this.path + "ewebeditpropopup.htm";
```

Object: "Popup Object"

Property: query

Description: Enter a query to pass parameters to the popup window.

NOTE

The popup.query property must not include the question mark (?) character.

Object: "Popup Object"

Property: windowFeatures

Description: The parameters passed to the standard JavaScript window.open() method.

To enable a feature (for example, scroll bars), include the keyword. To disable a feature, exclude the keyword. Separate each feature keyword by a comma, but *include no spaces* between parameters. A few of the possible features include:

- width=x, where x is the window width in pixels
- height=y, where y is the window height in pixels

- scrollbars: displays scrollbars
- status: displays the status bar
- resizable: the user can change the window size
- location: displays the location (or address) bar
- menubar: displays the menu bar
- toolbar: displays toolbar buttons

For more details on the JavaScript `window.open()` method, see a JavaScript reference.

Object: "Popup Object"

Property: `windowName`

Description: The name assigned to the popup window created by the standard JavaScript function `window.open()`.

Object: "Popup Object" and

Property: `editor`

Description: A reference to the **eWebEditPro+XML** ActiveX control. For example

`eWebEditPro.Editor1.pasteHTML("<HR>")` is equivalent to
`eWebEditPro.instances["Editor1"].editor.pasteHTML("<HR>")`

Read-only.

Object: "Instances Object"

Property: `elemName`

Description: The name of the field element that contains the editor content. This is typically the name specified when creating the editor.

Object: "Instances Object"

Property: `formName`

Description: The name or index of the form that contains this instance of the editor.

See Also: "Property: `elemName`" on page 174

Example:

```
function myOnEventHandler()
{
```

```
var objInstance = eWebEditPro.instances[eWebEditPro.event.srcName];
var strContent =
document.forms[objInstance.formName].elements[objInstance.elemName].value;
}
```

Object: "Instances Object"

Property: height

Description: The height of the editor assigned when created. Read-only.

Object: "Instances Object"

Property: html

Description: A string containing the HTML. To create the editor in a window other than the current one, set eWebEditPro.parameters.editorWindow to the name of the window. For example

```
<script>
frame2, document, open();
eWebEditPro.parameters.editorWindow="frame2";
eWebEditPro.create(...);
frame2.document.close();
</script>
```

To prevent the editor from writing the HTML to the window document, set eWebEditPro.parameters.writeDisabled to **true**.

For example,

```
<script>
eWebEditPro.parameters.writeDisabled="true";
var Obj Editor=eWebEditPro.create(...);
var strHTML=objEditor.html;
</script>
```

Object: "Instances Object"

Property: id

Description: The name of the **eWebEditPro+XML** editor element in the object (Internet Explorer) or embed (Netscape) tag. Typically not used. Read-only.

Object: "Instances Object"

Property: maxContentSize

Description: See "Property: maxContentSize" on page 168

Object: "Instances Object"

Property: name

Description: The name assigned to this instance of the editor when it was created. Read-only.

Object: "Instances Object"

Property: readOnly

Description: See "Property: readOnly" on page 170

Object: "Instances Object"

Property: receivedEvent

Description: This boolean value is **true** if an event has been received from the ActiveX control.

This property is used internally and is for reference only. It is not necessary for typical development.

Object: "Instances Object"

Property: status

Description: The status of this editor. (It has the same status value as the **eWebEditPro+XML** JavaScript object, but only applies to this instance of the editor). Do not change.

The status of the **eWebEditPro+XML** JavaScript object is described in "Property: status" on page 179.

Object: "Instances Object"

Property: type

Description: Indicates which type of editor was created on a page. Some values are listed below. You should not set this property -- consider it read-only.

activex - the editor implemented as an ActiveX control was created

textarea - a standard HTML textarea field was created instead of a full-featured editor

Example:

```

if ("activex" == eWebEditPro.instances[0].type)
{
    :
}
else if ("textarea" == eWebEditPro.instances[0].type)
{
    :
}

```

Object: "Instances Object"

Property: width

Description: The width of the editor assigned when created. Read-only.

Object: "Instances Object"

Property: {editor name}

Description: A reference to an instance of the **eWebEditPro+XML** ActiveX control.

See Also: ["Appendix A: Naming the eWebEditPro+XML Editor" on page 778](#)

Object: "eWebEditPro Object"

Property: actionOnUnload

Description: applicable to Internet Explorer only

Value	Description
EWEP_ONUNLOAD_SAVE (default)	When the Web page is unloaded, the content is saved to a hidden field on the Web page without prompting. The content is posted to the server only when the user clicks a Submit button.
EWEP_ONUNLOAD_NOSAVE	When the Web page is unloaded, the content is <i>not</i> saved to a hidden field.

Warning! *All changes since the last save are lost. For example, if the user presses the Back button, content in standard HTML elements is preserved, but any changes made in the **eWebEditPro+XML** editor are lost.*

Value	Description
EWEP_ONUNLOAD_PROMPT	<p>When the Web page is unloaded (except for submit), a dialog box prompts the user to click OK to save changes. (The dialog box text is defined by querySave in ewebeditpromessages.js).</p> <p>If the user clicks Cancel, another dialog box prompts whether to discard the changes or stay on the same Web page. (The dialog box text is defined by confirmAway in ewebeditpromessages.js).</p>

Object: "eWebEditPro Object"

Property: instances collection

Description: An array of in-line editor objects of type eWebEditProEditor or eWebEditProAlt (if the editor could not be created).

This array may be indexed by an integer (0 to instances.length-1) or by the name of an instance of an editor (for example, instances ["Editor1"]).

The **eWebEditPro+XML** Editor object has an editor property that provides a reference to the **eWebEditPro+XML** ActiveX control. For more information, see ["Event Handler Functions" on page 300](#).

Object: "eWebEditPro Object"

Property: installPopup

Description: If **true**, a window with the intro.htm page pops up.

See Also: ["Client Installation Pages" on page 297](#)

Return: boolean

Object: "eWebEditPro Object"

Property: isAutoInstallSupported

Description: If **true**, **eWebEditPro+XML** can be automatically installed. Currently, automatic installation is only supported on IE 5.0 or later.

If **false**, **eWebEditPro+XML** cannot be automatically installed. The client installation program is required to install **eWebEditPro+XML** on the client computer.

Return: boolean

Object: "eWebEditPro Object"

Property: isInstalled

Description: If true, eWebEditPro+XML is installed (or presumed installed). For Netscape, this indicates the Esker plug-in was installed. For IE, this indicates the editor is installed.

If false, eWebEditPro+XML is supported in this environment but needs to be installed on the client. By default, a message will appear prompting the user to install the client software.

Return: boolean

Object: "eWebEditPro Object"

Property: isSupported

Description: If true, eWebEditPro+XML is supported in this environment. It may not be installed yet.

If false, eWebEditPro+XML cannot run in this environment.

Return: boolean

Object: "eWebEditPro Object"

Property: parametersobject

Description: An object of type eWebEditProParameters containing the default set of parameters used when creating an instance of the editor or button.

To edit the default values set for the parameters, edit the ewebeditprodefaults.js file.

For more information, see ["Parameters Object" on page 7](#).

Object: "eWebEditPro Object"

Property: status

Description: Reflects the current state of eWebEditPro+XML.

Values:

- EWEP_STATUS_INSTALLED
- EWEP_STATUS_NOTLOADED
- EWEP_STATUS_LOADING
- EWEP_STATUS_LOADED
- EWEP_STATUS_SAVING
- EWEP_STATUS_SAVED
- EWEP_STATUS_NOTSUPPORTED

- EWEP_STATUS_NOTINSTALLED
- EWEP_STATUS_FATALERROR
- EWEP_STATUS_UNABLETOSAVE
- EWEP_STATUS_SIZEEXCEEDED
- EWEP_STATUS_NOTINSTALLED (The save was canceled because one or more popup editor windows is open)
- EWEP_STATUS_INVALID See Also: "Saving Invalid Documents" on page 625

Object: "eWebEditPro Object"

Property: upgradeNeeded

Description: If true, An older version **eWebEditPro+XML** is installed and needs to be upgraded. Currently only available with IE.

If false, **eWebEditPro+XML** is either the same or newer version, or could not be determined.

Return: boolean

Object: "eWebEditPro Object"

Property: Version

Description: The version of the control. It is a comma delimited list with this format:

Major Major, Minor Major, Major Minor, Minor Minor

Or

Version, Point Release, just 0, Revision

(The Major Minor value is not used, so it is always 0.)

Object: "eWebEditPro ActiveX Control Object"

Syntax: strData = [form!]ewebeditpro3.Version

Remarks

The Major Minor value of 0 is in the format because of the agreed upon format for software module versions. If comparing versions, the string must be parsed and each item converted to an integer.

Examples

Displays the control version:

```
function ShowVersion()
{
    alert(testItem1.Version);
}
or
alert(eWebEditPro.instances.MyContent1.editor.version);
or
alert(eWebEditPro.instances.MyContent1.editor.getPropertyString("version"));
```

Property: editorName

Valid in popup pages opened using eWebEditPro.openDialog(), this property holds the name of the editor that opened the popup. To access the instance JavaScript object associated with editorName, use getOpenerInstance().

Example

```
document.write("The page was opened by editor: " + eWebEditProUtil.editorName);
```

Object: "eWebEditProUtil Object"

Property: queryArgs

The array of URL query string parameters passed to the page. If the page is used in a frame, the queryArgs[] array holds the URL parameters of the topmost window.

Example

```
var whichFormElement = eWebEditProUtil.queryArgs["formelement"];
```

Object: "eWebEditProUtil Object"

Property: languageCode

The language code of the browser. If the language is not one of the known translated languages for the editor's menus and dialogs, this property is an empty string, that is, "". The languages for the editor's menus and dialogs are listed below.

Code	Language
ar	Arabic
da	Danish
de	German
es	Spanish
fr	French
he	Hebrew
it	Italian

Code	Language
ja	Japanese
ko	Korean
nl	Dutch
pt	Portuguese
ru	Russian
sv	Swedish
zh	Chinese (simplified)
zh-tw	Chinese (traditional)

Object: "eWebEditProUtil Object"

Property: editorGetMethod

Description: Lets you save either the body only or the entire HTML document from the editor. You can set this method in ewebeditprodefaults.js by editing this line:

```
this.editorGetMethod = "getBodyHTML"; // "getBodyHTML" or "getDocument"
```

Or, you can set this method directly in the Web page that calls the editor using the following JavaScript:

```
eWebEditPro.parameters.editorGetMethod = "value"
```

The possible values are **getBodyHTML** (see "Method: getBodyHTML" on page 83) and **getDocument** (see "Method: getDocument" on page 85).

Object: "Parameters Object"

Master List of Events

Event: ClickTag

Occurs when: The user clicks one of the following fields types on a Data Design screen.

- checkbox

- plain text
- select list
- button data fields in data entry mode

Object: "XML Object"

Parameters

Parameter	Type	Description
text	String	The name of the data field tag

Event: EditCommandComplete

Occurs when: This event notifies the client application or script that a user edit command has completed.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
CommandName	String	The command that has completed.

Remarks: This is an informational event for a client application, which may want to keep a log or look for certain commands to trigger certain functions.

Event: EditCommandStart

Occurs when: This event notifies the client application or script that a user edit command has started.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
CommandName	String	The command that has started.

Remarks: This is an informational event for a client application, which may want to keep a log or look for certain commands to trigger functions.

Event: EditComplete

Occurs when: This notifies the client application or script that an editing session has completed.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ImageName	String	The name of the image. See Also: " Image Names " on page 588
SaveFileName	String	The file name to which the image was saved. This includes the path and file extension.

Remarks: A user may have decided to complete the edit session, or an application may have closed WebImageFX.

Event: ImageError

Occurs when: This event notifies a client application or script that an error has occurred.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ErrorID	Long	A numeric value describing the error.
ErrorDesc	String	A string value describing the error.
ImageName	String	The name of the image that caused the error. See Also: " Image Names " on page 588 If a file could not be loaded or downloaded, this parameter lists the failed file. If the error is due to an initialization problem, this is an empty string.
Command	String	The command that was executed and caused the error. If there was no command, this is an empty string.

Remarks: See "[Method: ErrorDescription](#)" on page 78 to learn how errors are reported.

Event: LoadingImage

Occurs when: This event notifies a client application or script that an image file has loaded.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ImageName	String	The name of the image. <i>See Also:</i> "Image Names" on page 588 If the image is new, this is the name under which it is saved.
SaveFileName	String	The path and name of the file that contains changes for the image. If the image is remote, this is a temporary file name.
OldImageName	String	If a new image replaces an old image, this is the old image's name. If there is no old image, this is an empty string. <i>See Also:</i> "Image Names" on page 588
OldSaveName	String	The name of the image file being replaced. If the image is a remote image, this is a temporary file name. If there was no previous image, this is an empty string.

Remarks: When a user decides to edit a file, this event is called *before* an image is replaced or a new image is created. As a result, a client application or script can extract information about an image that is being replaced.

All functionality, such as producing HTML, works on the old image. The functionality against the new image is only available after this event completes.

This event is called even when a client application or script calls the EditFile or EditFromHTML methods.

Event: SavingImage

Occurs when: Before the current image is saved to the local file system.

Object: "Image Editor Object"

Parameters

Parameter	Type	Description
ImageName	String	The name of the image being saved. See Also: "Image Names" on page 588
SaveFileName	String	The path and file name of the saved image. If the image is remote, this is a temporary file name.

Remarks: When a file is saved, all changes are referenced in the name of the saved file. If content or a database is being updated, use the SaveFileName to reference the file.

The Image Name does not change when it is saved to a secondary file.

Event: ondblclickelement

Occurs when: Double-clicking on a hyperlink, applet, object, image, or table. See the ewebeditproevents.js file for an example of how to respond to this event.

oElement is a reference to the element object. The Variant returned is an HTML element object suitable for dynamic HTML (DHTML) scripting.

See a DOM reference for complete details on the element object. A few of the most useful common properties of the element object are listed below. Other properties are dependent on the type of element.

- tag name - the element's tag. For example,
oElement.tagName+”oElement.tagName+”;

NOTE

The plus sign (+) converts the tag name to a string.

- outerHTML - the entire HTML text of the element including the tag.

Object: "eWebEditPro ActiveX Control Object"

Event: onexeccommand

Occurs when: After a toolbar button is pressed, a drop-down or context menu (right-click menu) option is selected. This event can also be sent programmatically.

Object: "eWebEditPro ActiveX Control Object"

Parameters

ByVal strCmdName As String - The command that the user action executes

ByVal strTextData As String - Text associated with the command (typically not used)

ByVal IData As Long - Numeric data associated with the command (typically not used)

The IData parameter does not reflect the index of the list box item. Instead, it only returns the data assigned to the item.

If, in the processing of the command notification, you need the index of the selected item, use

```
objCommand.getPropertyInteger( "CmdIndex" )
```

Event: onfocus

Occurs when: The editor gains the focus. onfocus is a standard DHTML event.

Object: "eWebEditPro ActiveX Control Object"

WARNING!

This event does not work with Netscape or Firefox.

See Also: <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/events.asp>

Event: onblur

Occurs when: The editor loses the focus. onblur is a standard DHTML event.

Object: "eWebEditPro ActiveX Control Object"

WARNING!

This event does not work with Netscape or Firefox.

See Also: <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/events.asp>

Event: oncreate

Occurs when: The create method is invoked. If the event function returns false, the operation is aborted.

Object: "Event Object"

The eWebEditPro.event object properties:

The arguments passed to the create method. You can change the values of these properties in the oncreate event to alter the values used to create an instance of the editor.

Refer to the **create** method for a description of these arguments.

- name
- width
- height
- parameters

Event: oncreatebutton

Occurs when: The `createButton` method is invoked. If the event function returns `false`, the operation is aborted.

The eWebEditPro.event object properties:

The arguments passed to the `createButton` method.

You can change the values of these properties in the **oncreatebutton** event to alter the values used to create an instance of a popup button to the editor.

Refer to the `createButton` method for a description of these arguments.

- `buttonName`
- `elementName`
- `parameters`

Object: "Event Object"

Event: onbeforeedit

Occurs when: The `onbeforeedit` method is invoked. If the event function returns `false`, the operation is aborted.

The `onbeforeedit` method is called when the user clicks the button created by the `createButton` method.

The eWebEditPro.event object properties:

The argument passed to the `edit` method.

You can change the value of this property in the **onbeforeedit** event to change which instance of the editor is opened in the popup window.

Refer to the `edit` method for a description of this argument.

Object: "Event Object" and "eWebEditPro Object"

Event: onedit

Occurs when: After the popup window closes.

The eWebEditPro.event object properties:

Indicate which popup editor just closed.

- `elementName` - The name of the element that was just edited
- `popup` - A reference to the popup object

Object: "Event Object" and "eWebEditPro Object"

Event: onbeforeload

Occurs when: The load method is invoked. If the event function returns false, the operation is aborted.

The eWebEditPro.event object properties:

Undefined

Object: "Event Object", "Instances Object" and "eWebEditPro Object"

Event: onbeforesave

Occurs when: The save method is invoked. If the event function returns false, the operation is aborted.

The eWebEditPro.event object properties:

Undefined

Object: "Event Object", "Instances Object" and "eWebEditPro Object"

Event: ontoolbarreset

See Also: "Method: addEventHandler" on page 50 and "Reacting to the Initialization of a Toolbar" on page 303

Occurs when: The editor's toolbar is initialized or reset. Previously, you handled the toolbarreset command in the eWebEditProExecCommand() event handler function. Although that method still works, the preferred method is to add an event handler. For example:

```
eWebEditPro.addEventHandler("ontoolbarreset", "loadStyleSheet(this.event.srcName)");  
eWebEditPro.create("MyContent1", "100%", 400);
```

Object: "Event Object"

Event: onsave

Occurs when: The save method is complete. If the event function returns a boolean value (true or false), the save method returns the value.

A false value can be used to cancel leaving the page in some browsers. The save method is called when the page is unloaded, that is, in the document's onbeforeunload (IE only) or the onunload event, and also on the onsubmit event.

Note that the onsubmit event is not fired when the form's submit method is called. It only occurs when the user clicks the submit button.

If you are manually calling the submit method, also call the eWebEditPro.save method. The save method is not called on the onunload event if the window.eWebEditProUnloadHandled property is set true prior to calling the create method.

You can also prevent **eWebEditPro+XML** from copying content to the hidden field when the onsubmit event fires. To do this, set the `document.yourFormsName.eWebEditProSubmitHandled` property to **true** prior to calling the create method.

The eWebEditPro.event object properties:

Undefined

Object: "Event Object", "Instances Object" and "eWebEditPro Object"

Event: onload

Occurs when: The load method is complete.

The load method is called when the page is loaded, that is, in the document's onload event. The load method is not called if the `window.eWebEditProLoadHandled` property is set to **true** prior to calling the create method.

Object: "Event Object", "Instances Object" and "eWebEditPro Object"

The eWebEditPro.event object properties:

Undefined

Event: onready

Occurs when: It is safe to send commands or access the Media File Object.

NOTE

The preferred way to set an onready event handler is:

`eWebEditPro.addEventHandler("onready", your_event_handler);`
For more information, see "[Method: addEventHandler](#)" on page 50.

For example

```
eWebEditPro.onready = "initTransferMethod(eWebEditPro.event.srcName)";  
  
function initTransferMethod(strEditorName)  
{  
    eWebEditPro[strEditorName].MediaFile().setProperty("TransferMethod", "mediamanager.asp");  
}
```

Object: "Event Object" and "eWebEditPro Object"

The eWebEditPro.event object properties:

- **type** - Ready

- **srcName** - The name of the instance of the editor that is the source of the current event.

Event: onerror

Occurs when: An error occurs because the save method failed. Inspect the status property to determine the cause of the error. See Also: "Property: status" on page 179

Object: "Event Object", "Instances Object" and "eWebEditPro Object"

The eWebEditPro.event object properties:

Provide information about the source and cause of the error.

- **source** - The method that caused the error. For example, "load" if the load method failed.
- **name** - The name of the editor, where name is the argument passed to the create method.
- **instance** - A reference to the instance object.
- **reason** - Only available if save method failed because document is invalid - a numeric error code describing the reason for failure; 0 means no error occurred
- **message** - Only available if save method failed because document is invalid - a text message describing why validation failed.

You can localize the invalidContent message. See "Modifying the Language of eWebEditPro+XML" on page 265.

Event: eWebEditProReady

Occurs when: It is safe to send commands or access the Media File Object.

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

- type = "ready"
- srcName = name of the editor that is ready

See Also: "Appendix A: Naming the eWebEditPro+XML Editor" on page 778.

Example

```
eWebEditPro.onready = "initTransferMethod(eWebEditPro.event.srcName)";  
  
function initTransferMethod(strEditorName)  
{  
    eWebEditPro[strEditorName].MediaFile().setProperty("TransferMethod", "mediamanager.asp");  
}
```

Event: eWebEditProExecCommand

NOTE	Ektron recommends using the <code>eWebEditProExecCommandHandlers</code> array instead of this function. See "The <code>eWebEditProExecCommandHandlers</code> Array" on page 301.
-------------	--

Occurs when: After an internal command is executed, or when an external command should be executed. That is, when a toolbar button is pressed or a command is selected, such as on the context menu or dropdown list on the toolbar.

Writing the function `eWebEditProExecCommand` is the preferred way to add custom commands, rather than modifying `onExecCommandDeferred` or `onExecCommandHandler` in `ewebeditproevents.js`.

Return **true** to allow the default external commands to run.

Return **false** to prevent default external commands from running.

Internally handled commands are executed prior to this event's firing.

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

sEditorName - the name of the version of **eWebEditPro+XML** whose command was executed (for example, "MyContent1").

To access the **eWebEditPro+XML** methods, use
`eWebEditPro.instances[sEditorName].editor`.

See Also: "[Appendix A: Naming the eWebEditPro+XML Editor](#)" on page 778.

strCmdName - a string containing the command, for example, "conduct"

strTextData - a string that may contain text data related to the command.
Typically not used.

IData - a long integer value that may contain numeric data related to the command. Typically not used. The **IData** parameter does not reflect the index of the list box item. Instead, it only returns the data assigned to the item.

If, in the processing of the command notification, you need to retrieve the index of the selected item, use

```
objCommand.getPropertyInteger( "CmdIndex" ).
```

Event: eWebEditProMediaSelection

Occurs when: You want to add your own media file handler. This event occurs when the picture button is pressed.

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

(**sEditorName**) **sEditorName** is the name of the **eWebEditPro+XML** editor whose command was executed.

To access **eWebEditPro+XML** methods, use
`eWebEditPro.instances[sEditorName].editor`.

See Also: "[The ewebeditpromedia File](#)"

Event: eWebEditProMediaNotification

Occurs when:

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

Event: eWebEditProDbClickElement

Occurs when: A user double-clicks a hyperlink, applet, object, image or table within the editor, unless a specific event handler for hyperlink, image or table is defined.

To add a double-click element handler, define a JavaScript function in your Web page to run as shown below.

```
eWebEditProDbClickElement(oElement)
{
    return true or false
}
```

The **eWebEditProDbClickElement** function runs when certain elements are double-clicked. It may be easier, however, to define the applicable handler function for a specific object. The **hyperlink**, **image**, and **table** element objects have their own functions that run when they are double-clicked.

The default event handlers are defined in the **ewebeditproevents.js** file.

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

(oElement)

oElement is a reference to the HTML element that was double-clicked. Return

- **true** to allow the default external commands to run
- **false** to prevent them from running

Note that internally handled commands will have been executed prior to this event firing.

Event: eWebEditProDbClickHyperlink

Occurs when: A user double-clicks a hyperlink. The default hyperlink event handler is defined in the **ewebeditproevents.js** file.

Object: "ewebeditproevents Object"

The eWebEditPro.event object properties:

(oElement)

oElement is a reference to the HTML element that was double-clicked. Return

- **true** to allow the default external commands to run
- **false** to prevent them from running

Note that internally handled commands will have been executed prior to this event firing.

Event: eWebEditProDbClickImage

Occurs when: A user double-clicks an image.

Object: "eWebEditProEvents Object"

The eWebEditPro.event object properties:

(oElement)

oElement is a reference to the HTML element that was double-clicked. Return

- **true** to allow the default external commands to run
- **false** to prevent them from running

Note that internally handled commands will have been executed prior to this event firing.

Event: eWebEditProDbClickTable

Occurs when: a user double-clicks a table.

Object: "eWebEditProEvents Object"

The eWebEditPro.event object properties:

(oElement)

oElement is a reference to the HTML element that was double-clicked. Return

- **true** to allow the default external commands to run
- **false** to prevent them from running

Note that internally handled commands will have been executed prior to this event firing.

Commands

Commands define standard editor actions, such as changing the editor from WYSIWYG to Source View mode.

eWebEditPro+XML provides two types of commands:

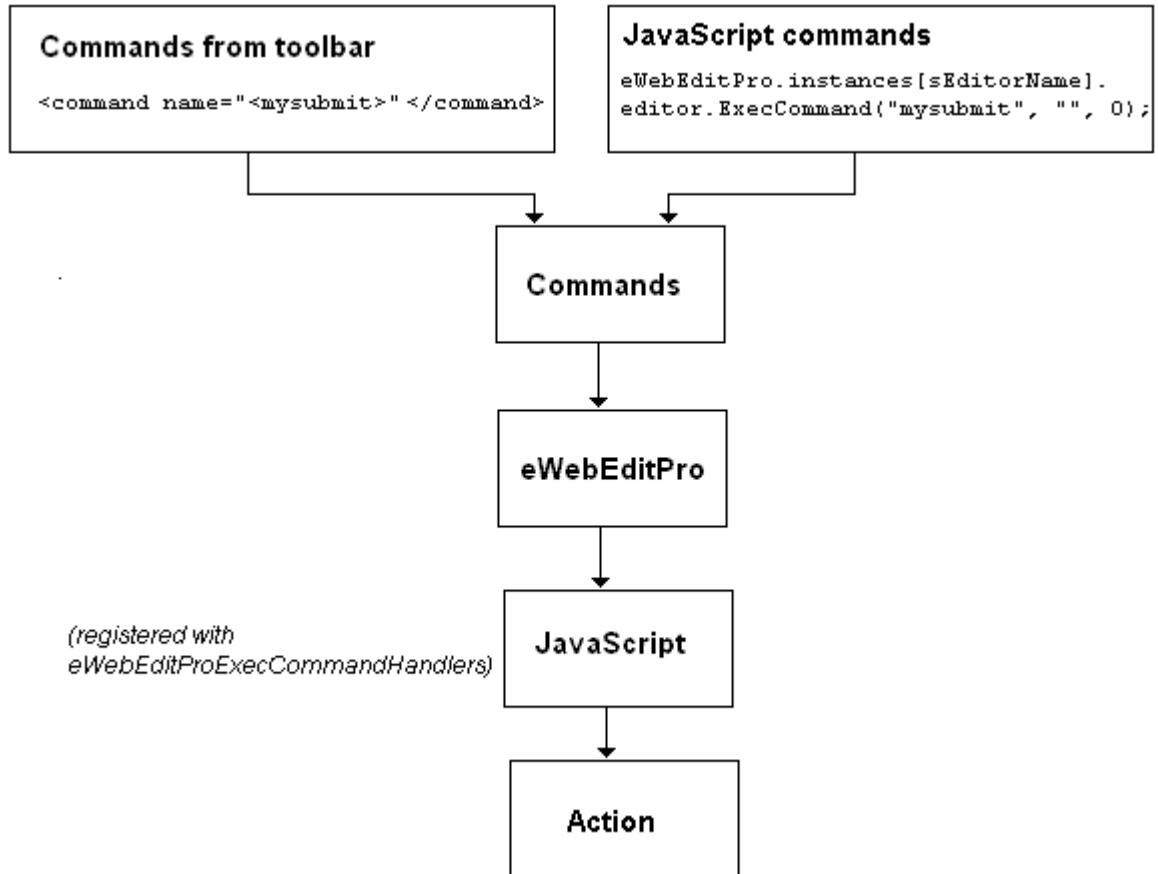
- Commands executed when a user clicks a toolbar button or menu option. Each **eWebEditPro+XML** feature has several user-executed commands. For example, the standard feature contains basic editing commands, such as cutting and pasting text.
- Commands that execute programmatically using JavaScript. For example, you want the editor to load in HTML view mode, rather than WYSIWYG. In this case, you add to the page that hosts the editor the command *cmdviewashtml*, which executes when the editor loads.

Programmatically-executed commands are explained in "[Using JavaScript to Send Commands](#)" on page 197.

Whether commands are initiated by a user or a script, there are two kinds of commands, standard and custom. See "[Sources of Commands](#)" on page 196.

How Commands are Processed

As shown in the illustration below, commands can be initiated from the toolbar or from a script. In both cases, the command is passed to the editor then to the JavaScript, which executes the action.



See Also: "Using JavaScript to Send Commands" on page 197

Sources of Commands

Regardless of what triggers a command, they have two sources.

Command source	Description	For more information, see
Standard	Supplied with eWebEditPro+XML	"Standard Commands" on page 199
Custom	You create them to extend the standard capabilities	"Custom Commands" on page 215

Using JavaScript to Send Commands

The following sample code illustrates how commands flow between JavaScript and eWebEditPro+XML.

The command in the sample, `mysubmit`, is registered using the `eWebEditProExecCommandHandlers` array.

In the sample, a button on the form is clicked to initiate the `mysubmit` command. The command is sent through the editor and received in the JavaScript.

NOTE

Instead of defining an external button, the command could have been assigned to a toolbar button or menu option.

```
<html>
<head>
<script language="JavaScript1.2" src="ewebeditpro.js"></script>
</head>
<body>
<form name="frmeditor1" method="post">
<script language="JavaScript1.2">
eWebEditProExecCommandHandlers[ "mysubmit" ] = function(sEditorName, strCmdName,
strTextData, lData)
{
    alert("Submitting the form.");
    document.forms["frmeditor1"].submit();
}
</script>
<input type="button" value="Run the mysubmit Command"
      onClick="eWebEditPro.instances['MyContent1'].editor.ExecCommand('mysubmit', '',1)">
<br />
<input type=hidden name="MyContent1" value=&lt;p&gt; Hello World &lt;/p&gt;" />
<script language="JavaScript1.2">
if (typeof eWebEditPro == "object")
{
    eWebEditPro.create("MyContent1", "100%", 300);
}
</script>
</form>
</body>
</html>
```

See Also: "How Commands are Processed" on page 195; "The eWebEditProExecCommandHandlers Array" on page 301

Sending Commands Programmatically

To send commands to the editor programmatically, use the `ExecCommand` method. Here is its format.

```
ExecCommand(strCmdName, strTextData, lData)
```

Here is how to access this method:

```
eWebEditPro.instances["EditorName" or index].editor.ExecCommand(...)
```

Examples:

```
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdcopy", "", 0);  
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdfontcolor", strColorValue, 0);
```

The ExecCommand method raises the onexeccommand event. See "[Event: onexeccommand](#)" on page 186.

Standard Commands

Standard Command Guidelines

Within the XML configuration data, a command's format is:

```
<command name="commandname" [enabled="true|false"]...>  
</command>
```

or

```
<cmd name="command name" .....>
```

Standard commands follow these guidelines.

- The element name is *command* or *cmd*.

NOTE

cmd is an abbreviated version of *command*. It reduces the time required to load configuration data. *<cmd>* has fewer attributes than *<command>*. Use *<command>* for more complex commands, such as dropdown lists. For a list of command element attributes, see "command" on page 336 and "cmd" on page 339.

- The element defines a specific command that a feature supports.
- A command's name
 - is defined in the *name* attribute of the command element
 - must follow the XML standard for names
 - must be unique within the configuration data
 - if supplied with **eWebEditPro+XML**, must begin with *cmd*. Custom commands must begin with *js*.
- A command's definition specifies its image, style, caption, and tooltiptext.

NOTE

When customizing a menu, users can assign any enabled command to a button even if the command is not assigned to any standard button.

List of Standard Commands

The following table lists the commands available within the standard feature of the configuration data.

See Also: "Commands" on page 195

- "Table Commands" on page 207
- "Special Character Commands" on page 212
- "Form Elements" on page 388 for forms toolbar commands

- "Commands Unique to WebImageFX" on page 593
- "Custom Tag Commands" on page 705

Command Name	Function
cmdabout	Displays the About eWebEditPro+XML screen
cmdabspos	<p>Absolutely positions an image, table or div tag. See Also: Ektron Knowledge Base article http://www.ektron.com/developers/ewebeditprokb.cfm?id=779</p>
cmdbackcolor	<p>Launches the Windows palette box. When the user selects a color, it is applied as a background to the selected text. For example:</p> <p>This text has a gray background.</p>
cmdbackcolorvalue	<p>Lets you create a list of background colors that can appear as a dropdown list on the toolbar. You can include the color "transparent."</p> <p>For more information, see "The cmdfontcolorvalue Command" on page 207.</p>
cmdbold	Applies bold to selected text
cmdbookmark	Applies a bookmark to selected text
cmdbullets	Assigns bullets to selected text
cmdcenter	Center justifies selected text
cmdchr	Instructs eWebEditPro+XML to insert a special character at the current cursor location. The text parameter (¡ through ÿ) identifies the character to insert.
cmdchr160	Inserts non-breaking space character ()
cmdchr169	Inserts copyright character (©)
cmdchr174	Inserts registered trademark character (™).
cmdclean	Cleans the HTML code (see "Cleaning HTML" on page 393)
cmdcopy	Copies selected text into the copy buffer
cmdcut	Deletes selected text and places it in the copy buffer

Command Name	Function
cmddelete	Deletes selected text
cmdextchars	<p>Display extended character dropdown list.</p> <hr/> <p>Note: This is not a command, but rather the name of a button that brings up a popup dialog. When an item is selected, the item's command is sent. This command is not used by itself. The configuration attaches it to a popup menu. If this command is sent by itself, nothing happens.</p> <hr/>
cmdfind	Launches the Find dialog box
cmdfindreplace	Launches the Find & Replace dialog box or performs operation if arguments are specified. See Also: "Find and Replace and Find Next Commands" on page 209
cmdfindnext	<p>From the current cursor location, searches for the next occurrence of the text in the Find What field of the Find & Replace dialog box.</p> <p>See Also: "Find and Replace and Find Next Commands" on page 209</p>
cmdfontcolor	<p>Launches the Windows palette box. When the user selects a color, it applies to the selected text.</p> <p>You can also assign a color using JavaScript. To do this, send the <code>cmdfontcolor</code> command to the editor.</p> <p>For example, a toolbar button opens a Web page that displays the 216 Web safe colors. When the user clicks a color, your page needs to execute JavaScript (shown below) that assigns the color.</p> <pre>eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdfontcolor" , strColorValue, 0);</pre> <p>In this script, <code>strColorValue</code> is a valid HTML color string, such as, "lightgreen" or "#C0C0C0". For more information on valid color strings, see "Describing Colors" on page 208.</p> <p>To display the standard Windows color palette, pass an empty string as the color value. Here is an example.</p> <pre>// Open standard windows color palette eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdfontcolor" , " " , 0);</pre>
cmdfontcolorvalue	Lets you create a list of colors that can appear as a dropdown list on the toolbar. For more information, see "The cmdfontcolorvalue Command" on page 207 .
cmdfontname	Assigns selected font name to selected text (see "fontname" on page 380)
cmdfontsize	Assigns selected font size to selected text (see "fontsize" on page 381)

Command Name	Function
cmdheaderlevel	Assigns selected header level to selected text (see "headings" on page 382)
cmdhr	Inserts a horizontal line (that is, an <hr> tag)
cmdhyperlink	Lets you edit a hyperlink assigned to selected text. See Also: "Managing Hyperlink Dialogs" on page 445
cmdindentleft	Decreases the indentation of selected text
cmdindentright	Increases the indentation of selected text
cmditalic	Applies italic to selected text
cmdjustify	Justifies both left and right sides of a paragraph, so that the text is aligned to the left and right margins. To accomplish this, the editor adds additional spaces between words of the line.
cmdleft	Left justifies selected text
cmdlock	"Locks" an image or table in its current position. Users use the cmdabspos command to absolutely position the item
cmdltrblk	Displays editor content in left-to-right orientation. This command's function is the opposite of the cmdrtlblk command, which displays content in a right-to-left orientation. Most Western European languages are read left-to-right. Some Asian languages, such as Arabic, Hebrew and Farsi, are read right-to-left.
cmdmueditimage	Launches WebImageFX. See "WebImageFX" on page 577.

Command Name	Function
cmdmfuinsert	<p>Initiates the insertion of a selected image or file. It is the command sent when the insertMediaFile function is called. (See “Method: insertMediaFile” on page 94.) You only need to insert this command if the function cannot be called.</p> <p>The command uses the information already given to the Media File Object. The string parameter to the command should be empty. The long parameter is 1 if the file is local, and 0 if it is remote.</p> <p>The function <code>insertMediaFileDeferred</code> (called from <code>insertMediaFile</code>) demonstrates its use.</p> <pre data-bbox="400 494 1264 944"><code>function eWebEditProEditor_insertMediaFileDeferred (strSrcFileLocation, bLocalFile, strFileName, strFileType, nWidth, nHeight) { // Place the file information into the media file object. // This is used for the insertion of the HTML. var objMedia = this.editor.MediaFile(); objMedia.setProperty("IsLocal", bLocalFile); objMedia.setProperty("SrcFileLocationName", strSrcFileLocation); objMedia.setProperty("FileName", strFileName); objMedia.setProperty("FileType", strFileType); objMedia.setProperty("ImageWidth", nWidth); objMedia.setProperty("ImageHeight", nHeight); this.editor.ExecCommand("cmdmfuinsert", "", bLocalFile); }</code></pre>
cmdmfumedia	Launches the Picture dialog box. (For more information, see “ Managing Images ” on page 455.)
cmdmfuuploadall	Lets user manually perform the upload process. (For more information, see “ cmdmfuuploadall Command ” on page 523.)
cmdmath	Enables the math editor.
cmdmsword	Launches Microsoft Word. (For more information, see “ Editing in Microsoft Word ” on page 411.)
cmdnojustify	Removes align attribute (after the command is applied, the paragraph will be left aligned unless the direction is right-to-left)
cmdnumbered	Applies numbers to selected text

Command Name	Function
cmdopen	<p>Displays standard Open File dialog, which lets user select an HTML file to load into the editor.</p> <p>A user <i>cannot</i> drag and drop a file into the editor to open it -- he or she must click the toolbar button or menu option that launches the dialog.</p> <hr/> <p style="color: blue;">This dialog appears in the selected language of the operating system, which is not necessarily the language selected for the editor.</p> <hr/>
cmdpaste	Pastes the contents of the copy buffer
cmdpastetext	<p>Pastes the contents of the clipboard as plain text. That is, all HTML tags (including images) are ignored.</p> <p>This command is helpful when the user wants to eliminate the HTML formatting of the source text being copied.</p>
cmdpicture	Launches the Picture dialog box
cmdprint	Prints the contents of the editor
cmdprop	<p>Shows the properties dialog of objects like Flash.</p> <p>To use this command, insert an object into the editor content. Then, select it and press the Properties button to view its properties.</p>
cmdredo	Reverses the most recent undo command
cmdremoveformat	Removes all character formatting, such as bold, italic, superscript, and color from the selected text
cmdright	Right justifies selected text
cmdrtlblk	<p>Displays editor content in right-to-left orientation.</p> <p>See Also: “cmdltrblk” on page 202</p>
cmdrtledit	<p>Moves the scroll bar to the right side of the editor window. You would typically use this with left-to-right orientation.</p> <p>See Also: “cmdltrblk” on page 202</p>
cmdsaveas	<p>Displays the standard Save File dialog box, which lets the user save the editor content to an HTML file.</p> <hr/> <p style="color: blue;">This dialog appears in the selected language of the operating system, which is not necessarily the language selected for the editor.</p> <hr/>

Command Name	Function
cmdselectall	Selects all editor content
cmdselectnone	Removes selection so that nothing is selected
cmdselstyle	Displays a dropdown list of style classes. Users can select from the list to apply a style class to selected text. (See “ Implementing Style Class Selectors ” on page 439.)
cmdshowborders	Display or suppress table borders only if the table border size is set to zero (0). If the table border size is set to 1 or higher, borders display regardless of this button's state.
cmdshowdetails	<p>Display invisible elements. Certain HTML elements (such as paragraph tags and <code></code> tags) are normally invisible in WYSIWYG mode. To indicate these elements, this button displays small glyphs, or images, to mark their position in the document. The glyphs make it less likely for a user to change an element by mistake.</p> <p>Note: <code><A></code> tags do not appear when the cmdshowdetails button is depressed.</p>
cmdspellayt	<p>Turn on or off spell check as-you-type feature.</p> <p>See Also: “The Spellcheck Feature” on page 404</p>
cmdspellcheck	Begin manual spell check. See Also: “ The Spellcheck Feature ” on page 404
cmdstrike	Applies strikethrough to selected text
cmdsub	Applies subscript to selected text
cmdsups	Applies superscript to selected text
cmdunderline	Applies underline to selected text
cmdundo	Reverses the most recent command
cmdunlink	Removes a hyperlink assignment from selected text
cmdunstyle	<p>Removes all style information applied to selected text. (You apply styles using the <code>style</code> dropdown list.) For example</p> <p>Before <code><P class="note">This is initial content.</P></code></p> <p>After <code><P>This is initial content.</P></code></p>
cmdviewashtml	Displays the page content as HTML, XHTML, or XML source.

Command Name	Function
cmdviewasproperties	Displays a dialog box listing display preferences (font color, size and style) for “View as HTML” mode. The user can change the values if desired.
cmdviewaswysiwyg	Displays the page content as WYSIWYG (What You See Is What You Get).
cmdzabovetext	Moves selected object above text covering it, if the object and text both occupy the same position.
cmdzback	Moves selected object behind all others, if more than one object occupies the same absolute position
cmdzbackward	Moves selected object behind object that occupies the same absolute position
cmdzbelowtext	Moves selected object behind text that occupies the same position within the editor
cmdzforward	Moves selected object in front of the object covering it, if two objects occupy the same absolute position
cmdzfront	Moves selected object in front of all others, if more than one object occupies the same absolute position
cmdzorder	Launches menu of options for positioning objects and tables above or below each other
cmrdltredit	Moves the scroll bar to the left side of the editor window. You would typically use this with left-to-right orientation. See Also: “cmdltblk” on page 202
jstm	Insert trademark character
js508table	Displays the Section 508 Table Properties HTML popup. The name of the popup page is section508table.htm.
jshyperlink	Displays an external web page that administrators can configure, customize, and add their own list of legal links. The name of the popup page is hyperlinkpopup.htm. In contrast, the cmdhyperlink command stays within the editor and administrators cannot customize it. See Also: “cmdhyperlink” on page 202
jscomment	It inserts a custom comment tag into content. The HTML popup used is commentpopup.htm. See Also: “Implementing a Command that Inserts a Comment” on page 710

Table Commands

Command Name	Function
cmdinserttable	Opens table dialog box to insert a new table
cmdinsertrow	Inserts a row above the cursor
cmdinsertcolumn	Inserts a column to the left of the cursor
cmdinsertcell	Inserts a cell to the left of the cursor
cmddeleterows	Deletes selected rows
cmddeletecolumns	Deletes selected columns
cmddeletecells	Deletes selected cells
cmdmergecells	Merges selected cells into one cell
cmdsplitcell	Splits the selected cell into two cells
cmdtable	Displays a menu of options for creating and converting tables
cmdtableproperties	Opens the table properties dialog
cmdcellproperties	Opens the cell properties dialog

See Also: “[Managing Tables](#)” on page 371

The cmdfontcolorvalue Command

The cmdfontcolorvalue command lets you create a list of colors that can appear as a dropdown list on the toolbar. The user can choose a color from the list

and apply it to selected text. Below is an illustration of what the dropdown list might look like.



For each listchoice data item, enter a data value (in this case, a color) and a description.

Below is a sample of how you might set up this command. This sample is provided in the config.xml file.

```
<command name="cmdfontcolorvalue" style="list" enabled="true">
  <caption>Font Color</caption>
  <toolTipText>Font Color</toolTipText>
  <selections name="fontcolorlist" enabled="true" sorted="true">
    <listchoice command="noop">Select Color</listchoice>
    <listchoice data="#00FFFF">Aqua</listchoice>
    <listchoice data="#000000">Black</listchoice>
    <listchoice data="#0000FF">Blue</listchoice>
    <listchoice data="#FF00FF">Fuchsia</listchoice>
    <listchoice data="#808080">Gray</listchoice>
    <listchoice data="#008000">Green</listchoice>
    <listchoice data="#00FF00">Lime</listchoice>
    <listchoice data="#800000">Maroon</listchoice>
    <listchoice data="#000080">Navy</listchoice>
    <listchoice data="#808000">Olive</listchoice>
    <listchoice data="#800080">Purple</listchoice>
    <listchoice data="#FF0000">Red</listchoice>
    <listchoice data="#C0C0C0">Silver</listchoice>
    <listchoice data="#008080">Teal</listchoice>
    <listchoice data="#FFFF00">Yellow</listchoice>
    <listchoice data="#FFFFFF">White</listchoice>
  </selections>
</command>
```

Describing Colors

To describe a color, use the standard HTML hexadecimal color reference, #RRGGBB, where RR is the red value, GG is green, and BB is blue. Each segment can have a value between 00 (0) and FF (255). Black is #000000, and white is #FFFFFF.

Alternatively, you can enter the color as a decimal value. To do this, omit the leading pound sign (#) character, for example, 16711935.

Note that some browsers do not support all colors.

Assigning a Color Using JavaScript

You can send the cmdfontcolorvalue command using JavaScript, where the value is passed as the IData argument. You can also use the cmdfontcolor command to assign the color using a string instead of a numeric value. To learn how to do this, see “[cmdfontcolor](#)” on page 201.

Assigning a Text Value to the Selection

The text value between the listchoice tags appears in the dropdown list. The text can be the color name, such as **red**, in English or another language. The text can also describe the color's purpose, for instance, **Company Logo Blue** or **Keyword Color**.

See Also: “[Creating a List Item that Generates No Command](#)” on page 249.

Displaying a Palette of Colors

As you can with any list box, you can add a list item that executes a command. If you want to let users select a color from the Windows palette box, you could add the following item to the list:

```
<command name="cmdfontcolorvalue" style="list" enabled="true">
```

You could also use the sample code provided below to open a popup window that displays a palette of colors from which the user can choose. You need to add JavaScript to process the custom jswebcolors command. Notice that a command attribute is present rather than the data attribute.

```
<listchoice command="jswebcolors">Web Safe Palette</listchoice>
```

Find and Replace and Find Next Commands

The user or developer (using JavaScript) can search and replace text within the content. This function is initiated by two standard toolbar buttons.

Icon	Command Name	Description
	cmdfindreplace	See “ cmdfindreplace ” on page 201
	cmdfindnext	See “ cmdfindnext ” on page 201

When the user clicks the Replace button, the following dialog box appears.



TIP The older find dialog (WYSIWYG mode only) is still available by pressing **Ctrl+F** or by using the **cmdfind** command.

This section describes the following subtopics related to the find and replace and find next commands:

- [Using the Commands in JavaScript](#)
- [Flags](#)
- [Including a Slash in the Search String](#)
- [Learning How Many Times Search Text Was Replaced](#)
- [Arguments to cmdfind and cmdfindnext Commands](#)

Using the Commands in JavaScript

The **cmdfindreplace** command accepts the **ExecCommand** method's string data parameter. If the string data parameter does not start with a slash character (/), the editor searches for the specified text. For example:

```
objEditor.ExecCommand('cmdfindreplace','is',0);
```

If the string data parameter starts with a slash, it is interpreted and parsed to extract the search text, replacement text, and optional flags. The parameter follows this format:

/ search text / replacement text / flags (explained below)

For example (the string data parameter is in **red *italics***):

```
objEditor.ExecCommand('cmdfindreplace','/t/(tee)/i',0);
```

Flags

Abbreviation	Name	Description
g	global	Replace all occurrences within the content, regardless of which content is currently selected
s	selection	Replace all occurrences within the currently selected content
i	case-insensitive	Search regardless of case. Checks Match case . That is, "a" and "A" are considered the same.
q	quiet	Suppress message indicating the search is complete
u	up	Search in the up direction rather than down. Selects Up .
w	word	Search for whole word only. Checks Match whole word only .

Examples

Description	Example
Replace next occurrence of 't' with '(tee)'.	objEditor.ExecCommand('cmdfindreplace','/t/(tee)',0);
Replace next occurrence of 't' or 'T' with '(tee)'.	objEditor.ExecCommand('cmdfindreplace','/t/(tee)/i',0);
Replace all occurrences of 'n' with 'N' and don't display the 'done searching' message.	objEditor.ExecCommand('cmdfindreplace','/n/N/gq',0);
Replace all occurrences of 't' with 'T' within the current selection.	objEditor.ExecCommand('cmdfindreplace','/t/T/gs',0);
Replace all occurrences of the word 'is' with 'was', unless 'is' appears within a larger word.	objEditor.ExecCommand('cmdfindreplace','/is/was/gw',0);
Find the next occurrence of the search text as currently specified in the dialog. Same as 'cmdfindnext'.	objEditor.ExecCommand('cmdfindreplace','',0);

Description	Example
Find the next occurrence of the search text, but don't display the 'done searching' message.	objEditor.ExecCommand('cmdfindreplace','/q',0);
Find the previous occurrence of the search text as currently specified in the dialog.	objEditor.ExecCommand('cmdfindreplace','/u',0);

Including a Slash in the Search String

To include a slash character (/) in the search or replacement text, precede it by two backslash (\) characters. The example below replaces '/' with '//'.

```
objEditor.ExecCommand('cmdfindreplace','/\//\//\//\//g',0);
```

Learning How Many Times the Search Text Was Replaced

The integer data returned in the onexeccommand event indicates the number of times the search text was replaced. A value of zero (0) means the text was not found.

For example, this event function displays the number of times the find text was replaced.

```
eWebEditProExecCommandHandlers["cmdfindreplace"] = function
(sEditorName, strCmdName, strTextData, lData)
{
    alert(strCmdName + " (" + strTextData + ", " + lData + ")");
};
```

Arguments of cmdfind and cmdfindnext Commands

The arguments of cmdfind and cmdfindnext commands are ignored. The following code is the same as pressing the Find Next button.

```
objEditor.ExecCommand('cmdfindnext','','',0)
```

Special Character Commands

You can use the following commands to create toolbar buttons that let users insert special characters. For example, cmdchr128 inserts the Euro character (€), whose Microsoft Windows Extended Character Reference value is 128.

Note

These characters may not appear on older browsers.

To learn how to place a command on a toolbar button, see ["Adding a Toolbar Button" on page 237](#).

Character	Command name	Image key	Description
€	cmdchr128	euro	euro
„	cmdchr130	lsquor	low left single quote
ƒ	cmdchr131	fnof	function of, florin
„	cmdchr132	ldquor	low left double quote
...	cmdchr133	hellip	ellipsis
†	cmdchr134	dagger	dagger
‡	cmdchr135	ddagger	double dagger
%	cmdchr137	permil	per thousand sign
š	cmdchr138	sscaron	S caron
‘	cmdchr139	lsquo	left single angle quote
Œ	cmdchr140	oeoelig	OE ligature
Ž	cmdchr142	zzcaron	Z caron
‘	cmdchr145	lsquo	left single quote

Character	Command name	Image key	Description
,	cmdchr146	rsquo	right single quote
“	cmdchr147	ldquo	left double quote
”	cmdchr148	rdquo	right double quote
•	cmdchr149	bull	round filled bullet
—	cmdchr150	ndash	en dash
—	cmdchr151	mdash	em dash
TM	cmdchr153	trade	trademark sign
š	cmdchr154	scaron	s caron
›	cmdchr155	rsaquo	right single angle quote
œ	cmdchr156	oelig	oe ligature
ž	cmdchr158	zcaron	z caron
Ÿ	cmdchr159	yyuml	Y umlaut

Custom Commands

The easiest and quickest way to add functionality to **eWebEditPro+XML** is to add custom commands.

Both standard and custom commands are executed when

- a user clicks a toolbar button or menu option
- an `onexeccommand` event is raised. This event sends the command to a client application or a script defined in the external section of the configuration data.

See Also: "Commands" on page 195

Custom Command Guidelines

Custom commands must follow these guidelines.

- They follow internal command conventions (see "Standard Command Guidelines" on page 199).

NOTE

Standard commands begin with *cmd*. Custom commands begin with *js*.

- Toolbar buttons or menu options are created for the command as specified
- The command's functionality is defined in an external client application or script
- **eWebEditPro+XML**'s only response to a custom command is to raise an event
- It is the responsibility of the script/client to define the event functionality

Creating a Custom Command

To create a custom command, you have two choices. You can

- **define a custom JavaScript function** - use if the custom command appears on every editor on a page
- **write a custom JavaScript event function** - use if the custom command appears in only one editor, or the command behaves differently in one editor. For example, you want to detect the `onblur` event for the last editor on a page, but not for editors above it.

Each method is explained below.

Defining a Custom Function for All Occurrences of the Editor

To define a custom JavaScript function for all occurrences of the editor, follow these steps. An example is provided to illustrate the steps.

1. Add JavaScript to the page that displays the editor
2. Create tooltip and caption text
3. Add the button to the toolbar
4. Add the command to the list of commands

The Example Used to Illustrate this Procedure

These instructions use an example to explain how to follow them. The example assumes that

- you want to create a custom button
- the button contains an image of the world (). The image is a standard image, supplied with **eWebEditPro+XML**.
- pressing the button executes a custom JavaScript command whose internal name is `jsexcompany`
- the command inserts into the editor your company's name ("My World Com") followed by the registered trademark symbol (®). The `®` value is the registered trademark symbol.

Add JavaScript to the Page that Displays the Editor

1. Decide where to place the JavaScript (see “[Where to Write a Custom JavaScript Function](#)” on page 221). This example places it on the page that displays the editor.
2. Create a new HTML page and insert an instance of the editor. Or, modify an existing page that displays the editor. The page should reside in the directory with **eWebEditPro+XML**.

For example

```

<HTML>
<head>
<script language="JavaScript1.2" src="/ewebeditpro5/ewebeditpro.js"></script>
<title>eWebEditPro</title>
</head>
<body>
<script language="JavaScript1.2">
if (typeof eWebEditPro == "object")
{
    eWebEditPro.create("MyContent1", "100%", 300);
}
</script>
</form>
</body>
</HTML>

```

This example assumes your HTML page names the editor MyContent1. If the editor has a different name (check the line that begins with eWebEditPro.create), replace MyContent1 with your editor name.

3. Insert the eWebEditProExecCommandHandlers array between the body's <script> tags. You can insert it above or below the line that invokes the editor. See Also: "[The eWebEditProExecCommandHandlers Array](#)" on [page 301](#)

```
<HTML>
<head>
<script language="JavaScript1.2" src="/ewebeditpro5/ewebeditpro.js"></script>
<title>eWebEditPro</title>
</head>
<body>
<script language="JavaScript1.2">
eWebEditProExecCommandHandlers[ "jsextcompany" ] = function(sEditorName,
strCmdName,strTextData, lData)
if (typeof eWebEditPro == "object")
{
    eWebEditPro.create("MyContent1", "100%", 300);
}
</script>
</form>
</body>
</HTML>
```

NOTE If you are using Netscape 6 for editing, the parameters must be all lower case.

To learn about the array's arguments, see "[ExecCommandHandlersArray Parameters](#)" on [page 302](#).

4. Insert the custom JavaScript between curly brackets ({}). As described in Step 1, the JavaScript can be entered within this function or in a separate file. If the JavaScript is kept in a separate file in the same directory as the page you are editing, enter the file's name, for example:

```
{<script language="JavaScript1.2" src="yourfilename.js"></script >}
```

If the JavaScript file is kept in a different directory, enter a relative or full path to the file, for example:

```
<script language="JavaScript1.2" src="../jsfolder/yourfilename.js"></script >}
```

or

```
{<script language="JavaScript1.2" src="(http://www.mysite.com/jsfolder/yourfilename.js)"></script >}
```

5. Enter the following function, which includes the custom JavaScript, between the body's script tags.

```
<HTML>
<head>
<script language="JavaScript1.2" src="/ewebeditpro5/ewebeditpro.js"></script>
<title>eWebEditPro</title>
</head>
<body>
<script language="JavaScript1.2">
eWebEditProExecCommandHandlers[ "jsextcompany" ] = function(sEditorName,
strCmdName,strTextData, lData)
```

```

{
    if("jsextcompany" == strCmdName)
    {
        eWebEditPro.instances.MyContent1.editor.pasteHTML( "My World Com&#174;<BR>" );
    }
    return(true);
    if (typeof eWebEditPro == "object")
    {
        eWebEditPro.create("MyContent1", "100%", 300);
    }
}
</script>
</form>
</body>
</HTML>

```

6. If the JavaScript returns **true**, the external commands run. If it returns **false**, the commands do not run.

Note that internally handled commands are executed prior to this event's firing.

7. Save the HTML page.

Preventing Default JavaScript From Executing

To prevent a default JavaScript function within ewebeditproevents.js from executing, add the following code to the bottom of the function (that is, to have the function return “**false**”).

```
return false;
```

Create Tooltip and Caption Text

Tooltip text appears when the user hovers the cursor over a button (see “[toolTipText](#)” on page 358). Button caption text appears next to or instead of a button (see “[Caption](#)” on page 335).

For standard buttons, this text is stored in localization files and referenced in the custom command’s definition in the configuration data. For example, the standard cut button in config.xml looks like this:

```
<cmd name="cmdcut" key="cut" ref="cmdCut" />
```

The **ref** attribute’s value (in this example, **cmdcut**) is translated to a term in the appropriate localization file. For example, if you open the English localization file (locale0409b.xml) and search for cmdcut, you find **<ts id="cmdCut">Cut</ts>**. The phrase between the **<ts>** tags is the command’s tooltip and caption text.

To create tooltip and button caption text for your custom command, follow these steps.

1. Open your language’s localization file (see “[Locale Files](#)” on page 266).

NOTE

If your system is English, you must identify the English locale file in ewebeditprodefaults.js if you are creating a custom button.

2. Move to the <custom> section of the file, near the top.

```
<?xml version="1.0" encoding="UTF-8"?>
<locale version="2" product="eWebEditPro" xml:lang="en">
    <custom>
        <!--define custom 'ts' elements here with a unique id-->
    </custom>
    <standard>
```

3. Enter <ts> opening and closing tags for the custom command's tooltip and button caption text.
4. Insert the value of ref or localeref as the id attribute. To continue our example, the opening and closing tags might look like this:
`<ts id="jsextcompany"></ts>`
5. Between the tags, enter the command's tooltip and caption text. For example, if the caption and tooltip text should be "Company Name," that section of the localization file would look like this:
`<ts id="CmdCompany">Company Name</ts>`
6. Save and close the localization file.

Add a Button to the Toolbar

1. Open the config.xml file and find the interface section.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface>
```

2. Within the interface section, move to the toolbar menu that you want to modify. (See ["Finding a Toolbar Menu's Internal Name" on page 232](#).)
3. Create a new line within the menu tags where you want the new item to appear. Buttons appear on a toolbar menu in the sequence in which they appear within the menu definition.
4. Enter the custom command using the syntax <button command="command name" />. Replace command name with jsextcompany.
5. Enter the command name entered in the strCmdName argument of the eWebEditProExec command or the eWebEditProExecCommandHandlers array.

For example, if a toolbar menu has three buttons, **cut**, **copy** and **paste**, and you want to add a button after **paste**, move to the line following **paste** and add the button command, as illustrated below (red indicates text that you insert).

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="true">
    <caption localeRef="btnMainCap">Edit</caption>
    <button command="cmdcut"/>
    <button command="cmdcopy"/>
    <button command="cmdpaste"/>
```

```
<button command="jsextcompany" />
</menu>
```

Add the Command to the List of Commands

1. Within the features > external section of the config.xml file, enter the custom command using the syntax `<command name="command name">`. Replace `command name` with the name entered in the strCmdName argument of the `eWebEditProExecCommand` function or the `eWebEditProExecCommandHandlers` array. In this example, that name is `jsextcompany`.

2. On the line following the command, enter tooltip and caption text localization code for the command. To continue the example, you would enter

```
<features>
  <external>
    <cmd name="jsextcompany" ref ="CmdCompany">
```

3. If you want to assign an image to the toolbar button, decide whether you want to use a standard image (that is, one supplied with **eWebEditPro+XML**) or a custom image that you create.

- to specify the standard image "world", use this code:

```
<cmd name="jsextcompany" ref ="CmdCompany" key="world">
```

- to specify a custom image file named, for example `customcut`, located in folder to which **eWebEditPro+XML** is installed, use this code:

```
<cmd name="command name" image src="[eWebEditProPath]/customcut.gif" />
```

See Also: ["Changing the Image that Appears on a Toolbar Button" on page 242](#).

If you do not assign an image, the command's caption text appears on the button.

Testing the Page

After you finish these instructions, test the page to verify that it works as planned. When testing the page, you cannot simply double click the .html file. Instead, you must type the following url into the Web browser's address field:

```
http://localhost/ewebeditpro5/folder/filename.htm
```

For example, if the file is named `mytest.htm` and it is located in a folder named `ewebeditpro5`, enter this url into your browser:

```
http://localhost/ewebeditpro5/mytest.htm
```

Writing a Custom JavaScript Event Function for One or More Occurrences of the Editor

As an alternative to defining a custom function, you can write a custom JavaScript event handler function. To use this method of creating a custom command, follow these steps.

NOTE This is an abbreviated version of the detailed procedure explained in "Creating a Custom Command" on page 215. You should refer to that section for a step-by-step explanation of the following tasks.

1. Decide where you want to place the custom function (see "Where to Write a Custom JavaScript Function" on page 221).
2. Write the custom JavaScript function and place it in the location decided in Step 1.
3. On the page on which an editor is declared, enter

```
eWebEditPro.ExecCommandHandlers[ "name_of_your
custom_event_handler" ] = function(sEditorName,
strCmdName, strTextData, lData)
```
4. Insert the custom JavaScript.
5. During the onexeccommand event, the following event object properties are available.

Event Object Property	Description
eWebEditPro.event.srcName	The name of the editor <i>See Also:</i> "Appendix A: Naming the eWebEditPro+XML Editor" on page 778
eWebEditPro.event.cmdName	The name of the command that the toolbar button executes
eWebEditPro.event.textData	Text entered by the user (typically not used)
eWebEditPro.event.data	Data entered by the user (typically not used)

6. In the configuration data, assign the custom event handler to a toolbar button or menu. This procedure is documented in "Commands" on page 195.
7. In the configuration data, make the button or menu item available to users. This procedure is documented in "Adding a Toolbar Button" on page 237.

NOTE To dynamically change the behavior of a command, change the value of eWebEditPro.onexeccommand.

Where to Write a Custom JavaScript Function

There are three places where you can write a custom JavaScript function.

Where	Sample Code	Use this approach to implement the function in
On the page with the editor <i>See Also: "Appendix A: Naming the eWebEditPro+XML Editor" on page 778</i>	<pre><script language="JavaScript1.2"> enter JavaScript here eWebEditPro.create("MyContent1", 700,150); </script></pre>	a single occurrence of the editor or all occurrences of the editor on that page
In a JavaScript file, referenced on each page that uses it	<pre><script language="JavaScript1.2" src="yourfilename.js" </pre>	all occurrences of the editor on that page
In a JavaScript file, referenced among the standard JavaScript files in the ewebeditpro.js file. If you take this approach, the function is available to all pages without declaring it on each page.	<pre>var eWebEditProIncludes = ["ewebeditproevents.js", "ewebeditprodefaults.js", your_custom_JavaScript_file.js eWebEditProMsgsFilename, "ewep.js"];</pre>	all occurrences of the editor

Detecting When a Standard Command is Executed

You can use the `eWebEditProExecCommandHandlers` array to detect when a command has been executed. Then, within the function, you can add JavaScript to execute when the standard command executes.

For example, assume that whenever a user presses the underline toolbar button (U), a custom warning message appears and tells the user that readers of the Web page may mistake underlined words for hyperlinks.

Defining a Custom JavaScript Function

To define a custom JavaScript function, follow these steps.

1. Open a file (see "Where to Write a Custom JavaScript Function" on page 221).
2. Within the `<script>` tags, register the function using this code.

```
eWebEditProExecCommandHandlers["function name"] = function(sEditorName, strCmdName,
strTextData, lData)
```

To learn about the array's arguments, see "Parameter Requirements for Commands" on page 302.

3. Insert the JavaScript between the curly brackets ({}). Here is sample JavaScript for this example.

```
{  
    if (strCmdName=="cmdunderline")  
    {  
        alert("Underlined text may be confused with hyperlinks");  
    }  
}
```

- If the command is standard (begins with cmd), the JavaScript executes after the standard command is executed
- If the command is custom, the JavaScript executes before the command

To continue with the above example, when the user presses the underline button, the editor applies underline. Next, the eWebEditProExecCommandHandlers array executes the JavaScript from within the command, which displays the warning message to the user.

Using eWebEditPro+XML

Design and Implementation Guidelines

System Requirements

Browser for Editing	<ul style="list-style-type: none">● Microsoft Internet Explorer, version 5.0 or higher● Netscape Navigator, version 4.7x (with IE 5.0 or higher installed)● Netscape 6 (with IE 5.0 or higher installed) <p>Browsers must run under Microsoft Windows 98, NT, 2000 or later.</p>
Browser for Viewing	<ul style="list-style-type: none">● Microsoft Internet Explorer, version 3.0 or higher● Netscape Navigator, version 3.0 or higher● Netscape 6● Opera <p>or any other browser</p>
Dynamic Web Server Hardware	Based on the system requirements of the dynamic application server and/or Web server you are using.
Server Operating System	<ul style="list-style-type: none">● Windows NT Server, Windows 2000 Server● Windows 98, ME, 2000, XP with PWS● Sun Solaris● Linux● HP-UX● any other server OS

	<p>Client Hardware</p> <p>Any IBM-PC compatible system. Suggested minimum requirement: IBM compatible Pentium 166 with 64 Mb RAM.</p>
--	---

Maximum Size of Content

See "Property: maxContentSize" on page 168.

Placing More Than One Editor on a Page

You can easily place several editors on one Web page. To see an example of this, see the test drive on our Web site at <http://www.ektron.com/ewebeditpro3/testdrive/multiedit.htm>.

Samples

Ektron also provides sample code that shows how to put two editors on a Web page. The sample files reside in the folder to which you installed *ewebeditpro3\samples\your server platform\ multiedit.xxx* (xxx is the server extension, such as, asp, cfm, jsp).

For example,

C:\Inetpub\wwwroot\ewebeditpro3\samples\asp\multiedit.asp.

Memory Considerations

If you place more than one editor on a page, make sure that adequate memory resources are available. Windows '95 and '98 are not extremely reliable for memory. Windows 2000 is better able to manage the space needed to run several editors at once.

Recommendations

Ektron suggests running no more than 5 editors per page.

To place more than 5 editors on a page, we recommend using a popup button that opens one editor at a time. Alternatively, you could group them with a "next" key to bring up the next batch.

NOTE

There is no known limit to the number of popup editors.

eWebEditPro+XML Dataflow

This section describes how content flows from a database or file system on a Web server into the **eWebEditPro+XML** editor and then back to a Web server.

Integrating **eWebEditPro+XML** into a Web Page

The **eWebEditPro+XML** editor is a browser plug-in. It does not directly connect to a Web server or database. It may be used with any dynamic Web server and any database or no database at all.

eWebEditPro+XML includes JavaScript that facilitates integration into a Web page. Most of the details of moving the content are handled for you.

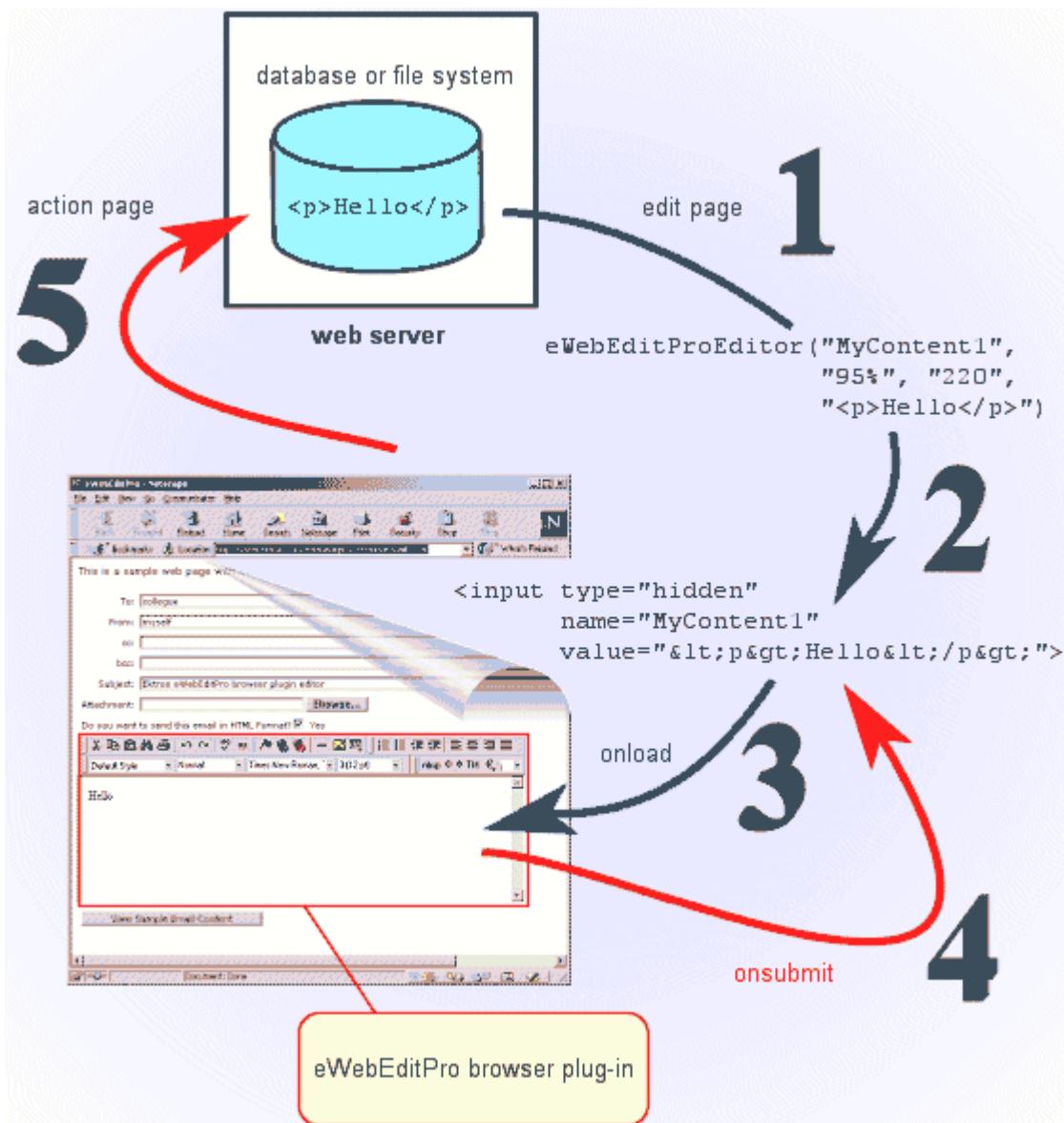
The **eWebEditPro+XML** editor replaces a standard HTML textarea field. The editor's content is stored in a standard HTML hidden text field. This means processing on the server-side is standard. In fact, if your Web application currently uses a textarea field that permits HTML tags, there is very little to change on the server-side--typically just a few lines of code to create the **eWebEditPro+XML** plug-in instead of a textarea field.

Integration files are provided for many platforms, including ASP, JSP, and many more. See the "["Integrating eWebEditPro+XML" on page 734](#)" for a complete list. If your platform language is not included, you can still easily integrate **eWebEditPro+XML** in a Web page using HTML and JavaScript. Many HTML samples are included.

See Also: "["Integrating eWebEditPro+XML Using JavaScript" on page 766](#)

Content Flow Diagram

The illustration below shows how content flows from the Web server to the editor and then back to the server. Text following the illustration explains it in more detail.



1. The Edit Page: Read Content

The Web page that hosts the editor reads the initial content from a database or a static file on the server. If a new content block or email message is being created, the initial content is an empty string.

You, as the developer, need to create this page (or use one of the samples provided). The editor plug-in is placed in a Web page by including the **eWebEditPro+XML** integration file for your platform and calling a function or

custom tag (depending on your language). The initial content (or empty string) is passed to the function or custom tag.

Language	Example
ASP	<% =eWebEditProEditor("MyContent1", "95%", "220", "<p>Hello</p>") %>
ASP.NET	<ewep:eWebEditProEditor id="MyContent1" runat="server" width="95%" height="220" Text="<p>Hello</p>"></ewep:eWebEditProEditor>
ColdFusion	<CF_ewebeditpro5 Name="MyContent1" Width="95%" Height="220" Value="<p>Hello</p>">
JSP	<%= eWebEditProEditor("MyContent1", "95%", "220", "<p>Hello</p>") %>
Perl	ewebeditpro::eWebEditProEditor("MyContent1", "95%", "220", "<p>Hello</p>");
PHP	<?php echo eWebEditProEditor("MyContent1", "95%", "220", "<p>Hello</p>") ?>
HTML and JavaScript	<input type=hidden name="MyContent1" value="<p>Hello</p>"> <script language="JavaScript1.2"> <!-- eWebEditPro.create("Description", "95%", "220"); //--></script>

For more information, see "[Integrating eWebEditPro+XML](#)" on page 734.

2. The Hidden Field

This step is done for you by the **eWebEditPro+XML** function or custom tag (unless you are just using HTML/JavaScript). The hidden field is a standard HTML hidden text field: `<input type="hidden">`. The hidden field must be placed within a standard HTML form. This is how the content is posted back to the server. The **eWebEditPro+XML** plug-in does not need to directly connect to the Web server to read or write the content.

The hidden field has the same name you assigned when creating the editor.

Note that, at this point, the content has been HTML encoded. For example, the "<" character is converted to < and double quotes ("") are converted to ". The **eWebEditPro+XML** function or custom tag performs this task for you.

3. The onload Event

This step is done for you by the **eWebEditPro+XML** JavaScript integration code. When the page's onload event fires in the client browser, the **eWebEditPro+XML** JavaScript integration code copies the content from the hidden field into the editor.

4. The onsubmit Event

This step is done for you by the **eWebEditPro+XML** JavaScript integration code. After a user modifies the content, he or she presses the submit button. When the

onsubmit event fires in the client browser, the **eWebEditPro+XML** JavaScript integration code copies the content from the editor to the hidden field.

5. The Action Page: Write Content

When a user presses the submit button, all HTML form field values are posted to the Web server, including the hidden field. The action page, which you as a developer write, processes the form field values.

Typically, in a content management application, the values are stored in a database. In an email application, an email message is sent.

Defining the Toolbar

When you look at **eWebEditPro+XML**, you see a box with one or more rows of buttons across the top, known as the toolbar. The following illustrates a section of the **eWebEditPro+XML** toolbar.



When you first load **eWebEditPro+XML**, the default toolbar appears. You can determine which items appear on the toolbar, and what happens when a user selects an item by modifying the configuration data.

Modifying Configuration Data

There are two ways that you can modify configuration data:

- *dynamically*, using JavaScript on the server, on the client, or both.
- *statically*, by editing an .xml file that stores the configuration data. The file's name is config.xml and, by default, is installed in the ewebeditpro5 directory.

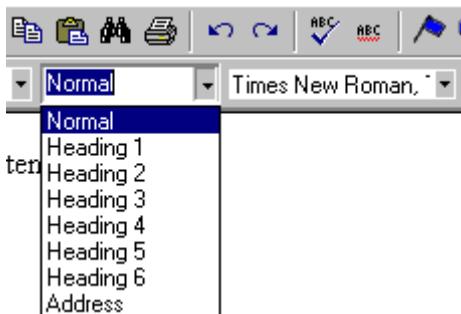
NOTE

If you use an XML editor to edit config.xml, Ektron supplies a corresponding schema file (config.xsd) that can validate config.xml. By default, the config.xsd is installed to the `ewebeditpro5` directory. Note that some validators might find errors when validating config.xml against config.xsd because some attributes have no value by default.

This section explains how to modify the toolbar and enact changes by editing the config.xml file. “[Dynamically Changing the Editor](#)” on page 250 explains how to change the toolbar using a script.

Toolbar Menus

The toolbar includes one or more toolbar menus. Each menu consists of one or more toolbar buttons or *dropdown lists* (illustrated below).



Here are key points about toolbar menus.

- A toolbar menu can reside on the same row with another menu. It can also continue to the next row if it cannot fit on a single row.
- Double vertical bars indicate the beginning of a new toolbar menu, as illustrated



- You *can* place all available items on a single toolbar menu. However, it's probably more efficient to create a few menus that provide a related set of functions, and activate those menus in the configuration data assigned to a user group.

Defining the eWebEditPro+XML Toolbar

There are two major aspects to defining the **eWebEditPro+XML** toolbar. You can define

- which toolbar menus appear on the toolbar, and the sequence in which they appear
- characteristics of each button and dropdown list

You can also create a popup menu that appears when the user presses a button. Finally, you can create custom commands as well add JavaScript that executes after a standard command is performed.

The following sections explain these procedures.

WARNING!

If you change the interface section of the configuration data, the user will not see the change if he or she has customized. For testing, to ensure that your changes appear, set the `allowCustomize` attribute of the interface element to `false` or change the name attribute of the interface element to a name not previously used. See Also: “[Letting Users Customize the Toolbar](#)” on page 315.

Determining Which Menus Appear on the Toolbar

When defining toolbar menus, you can perform the following tasks.

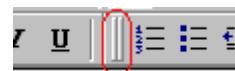
- Find a toolbar menu's internal name
- Add a custom toolbar menu
- Remove a toolbar menu
- Remove all toolbar menus
- Determine whether a toolbar menu can reside on a row with another menu or must appear on its own row
- If a toolbar menu does not fit on one row, determine if it should wrap around to the next row
- Create or edit the toolbar menu caption

NOTE If you want to add to or modify the buttons on a toolbar menu, see "Determining Which Buttons and Dropdown Lists Appear on a Menu" on page 237.

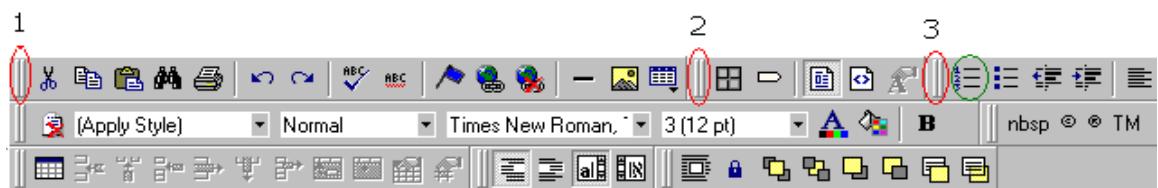
Finding a Toolbar Menu's Internal Name

Many procedures in this section require you to identify a toolbar menu's internal name. If you do not know a menu's internal name, follow these steps to learn it.

1. Count the number of the toolbar menu. Begin your count at the top left corner. (Remember that toolbar menus begin with double vertical bars.)



For example, in the following illustration, the numbers button ($\frac{1}{2}$) is part of the third toolbar menu.



2. Open the config.xml file and look for the first line that begins with <menu name (illustrated below in red).

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface name="standard" allowCustomize="false">
        <menu name="editbar">
```

This line is near the top of the file.

3. Within config.xml, all toolbar menus are indented the same distance from the left margin. Scroll down the list of menus until you see the menu that you identified in Step 1.

```

<menu name="editbar" newRow="false" showButtonsCaptions="true" wrap="false">
    first menu
        <button command="mnuEdit"/>
        <button command="cmdcut"/>
        <button command="cmdcopy"/>
        <button command="cmdpaste"/>
        <button command="cmdfind"/>
        <button command="cmdprint"/>
        <bar/>
        <button command="cmdundo"/>
        <button command="cmdredo"/>
        <bar/>
        <button command="cmdspellcheck"/>
        <button command="cmdspellaybt"/>
        <bar/>
        <button command="cmdbookmark"/>
        <button command="cmdhyperlink"/>
        <button command="cmdunlink"/>
        <bar/>
        <button command="cmdhr"/>
        <button command="cmdinfomedia"/>
        <button command="cmdtable" popup="tableP"/>
</menu>
<menu name="viewasbar" newRow="false" showButtonsCaptions="true" wrap="false">
    second menu
        <button command="mnuViewAs"/>
        <button command="cmdshowborders"/>
        <button command="cmdshowdetails"/>
        <bar/>
        <button command="cmdviewasmyimg"/>
        <button command="cmdviewashml"/>
        <button command="cmdviewasproperties"/>
        <button command="cmdnsword"/>
</menu>
<menu name="pformatbar" newRow="false" showButtonsCaptions="true" wrap="false">
    third menu
        <button command="mnuPFmt"/>
        <button command="cmdunnumbered"/>
</menu>
```

4. Now that you have identified the toolbar menu, you can add buttons to it or remove buttons from it.

Each task is described below.

Creating a Custom Toolbar Menu

You can create a custom toolbar menu and place any set of button commands on it.

While you can type in a toolbar menu definition, it is quicker to copy and edit one.

1. Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface>
```

2. Copy any toolbar menu definition (the text between the menu tags, <menu and </menu>).

Here is a typical toolbar menu definition.

```
<menu name="viewasbar" newRow="false" showButtonsCaptions="false" wrap="false">
    <caption localeRef="mnuViewAs"/>
    <button command="cmdshowborders"/>
```

```

<button command="cmdshowdetails" />
<bar/>
<button command="cmdviewaswysiwyg" />
<button command="cmdviewashtml" />
<button command="cmdviewasproperties" />
<button command="cmdmsword" />
</menu>

```

3. Move to the line in the config.xml file where you want the new menu to appear.
For example, if you want the custom menu to appear after the second standard menu, move to the line in config.xml following the description of the second standard menu.
4. Paste the menu definition you copied in Step 2.
5. Edit the toolbar menu's name and other attributes as appropriate. (Menu attributes are explained in “[menu](#)” on page 349.)
6. Remove buttons that should not be available (see “[Removing a Toolbar Button or Dropdown List](#)” on page 240).
7. Add new buttons as desired (see “[Adding a Toolbar Button](#)” on page 237 and “[Adding a Toolbar Button](#)” on page 237).
8. Users will see the new toolbar menu the next time they sign on.

Removing a Toolbar Menu

To remove a toolbar menu from the **eWebEditPro+XML** toolbar, follow these steps.

NOTE “[Overview of Configuration Data](#)” on page 319 explains how to edit the XML configuration file.

1. Identify the toolbar menu that you want to remove (see “[Finding a Toolbar Menu’s Internal Name](#)” on page 232.)
2. Within the definition of that toolbar menu, set the enabled attribute to false. Here is an example. (The text appears in red for illustration purposes only.)

```

<menu name="editbar" enabled="false" newRow="false"
      showButtonsCaptions="false"
      wrap="false"> <caption localeRef="btnMainCap">Edit</caption>
      <button command="cmdcut" />
      <button command="cmdcopy" />
      <button command="cmdpaste" />
      <button command="cmdfind" />
    <bar/>
</menu>

```

3. Users will not see the toolbar menu the next time they sign on.

Removing All Toolbars

To remove all toolbars from the **eWebEditPro+XML** editor, follow this step.

- Within the interface section of the configuration data, set the `visible` attribute to "false".

If the attribute does not appear, add it. Here is an example.

```
<interface name="standard" allowCustomize="false" visible="false">
```

For more information, see ["visible" on page 345](#).

Placing a Toolbar Menu on a Row with Another Menu

A menu's `newrow` attribute determines whether or not it can reside on the same row with another toolbar menu.

If the attribute is set to "false", the toolbar menu resides on the same row with



another menu.

If "true", a toolbar menu goes to the beginning of the next row.



The default value for this attribute is "true".

To change the `newrow` attribute, follow these steps.

- Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
  <!-- Valid positive values are: yes, true, 1 -->
  <!-- Valid negative values are: no, false, 0 -->
  <interface>
```

- Within the interface section, move to the definition of the toolbar menu that you want to modify. (See ["Finding a Toolbar Menu's Internal Name" on page 232](#).)
- Change the value of the `newrow` attribute. For example, assume that you want a toolbar menu to reside on the same row with the preceding menu. That section of the config.xml file would look like this.

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="false">
```

- Users will see the new toolbar menu arrangement the next time they sign on.

Determining if a Toolbar Menu Should Wrap to the Next Row

A toolbar menu's `wrap` attribute determines what happens when a menu's toolbar buttons extend beyond the right edge of the menu row.

If the attribute is set to "true", when the icons reach the right edge of the display area, they wrap to the next row.

If "false", the icons do not wrap to the next row. They are invisible until you move the toolbar menu bar to another row or drag it from the toolbar.

The default value for this attribute is "true".

To change the `wrap` attribute, follow these steps.

- Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface>
```

- Within the interface section, move to the toolbar menu that you want to modify. (See "Finding a Toolbar Menu's Internal Name" on page 232.)
- Change the value of the `wrap` attribute.

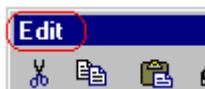
For example, if you change a toolbar menu's `wrap` attribute to "true", the line in config.xml looks like this:

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="true">
```

- Users will see the new toolbar menu arrangement the next time they sign on.

Creating or Editing the Toolbar Menu Caption

A toolbar menu caption only appears when the user drags the menu away from



the toolbar.

In this example, "Edit" is the menu caption.

```
<menu>
    <caption localeRef="mnueditcap">Edit</caption>
</menu>
```

To change the toolbar menu caption, follow these steps.

- Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface>
```

- Within the interface section, move to the toolbar menu whose caption you want to modify. (See "Finding a Toolbar Menu's Internal Name" on page 232.)
- Find the section of the toolbar menu definition that begins with `caption localeRef` (indicated in red below).

```
<menu name="pformatbar" newRow="false" showButtonsCaptions="false" wrap="false">
    <caption localeRef="mnuPFmt"/>
```

In this example, the locale code is `mnuPFmt`.

- Close the XML configuration file.

5. Open the localization file for the language of the editor. (By default, localization files reside in the ewebeditpro5 folder.) For example, if the editor language is English, open locale0409b.xml.
(For more information on localization files, see "Locale Files" on page 266.)
6. Within the localization file, find the locale code identified in Step 3. To continue the example in that step, you would find `mnuPFmt`.
7. Edit the text between the locale code tags. For example, change
`<mnuPFmt>Paragraph Format</mnuPFmt>` to
`<mnuPFmt>Format</mnuPFmt>`.
8. Users will see the new toolbar menu caption the next time they sign on.

Determining Which Buttons and Dropdown Lists Appear on a Menu

This section explains how to define the items that make up each toolbar menu. Menus are made up of toolbar buttons and dropdown lists. You can define the contents of toolbar menus in the following ways.

- Add a new toolbar button
- Add a new dropdown list
- Remove a toolbar button/dropdown list
- Rearrange the buttons/dropdown lists on a toolbar menu
- Add a space between two toolbar menu items
- Add a separator bar between two toolbar menu items
- Change the image that appears on a toolbar button
- Display or suppress button caption text
 - If you display caption text, you can define the alignment of the text on the button
- Translate button captions and tool tips to a foreign language

Adding a Toolbar Button

As explained in "button" on page 333, buttons or dropdown lists execute commands. Standard and custom commands are defined in the features section of the configuration data. (See "Commands" on page 195 for more details.)

To add a new button to a toolbar menu, follow these steps.

NOTE

Whether the button appears as a square with an icon or a dropdown list is determined in the command's `style` attribute, not when you create the button. For more information, see "Command Styles" on page 338.

1. Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
```

```
<!-- Valid positive values are: yes, true, 1 -->
<!-- Valid negative values are: no, false, 0 -->
<interface>
```

2. Within the interface section, move to the toolbar menu that you want to modify. (See "Finding a Toolbar Menu's Internal Name" on page 232.)
3. Move to the line within the menu tags where you want to add the new item. Buttons and dropdown lists appear on a toolbar in the sequence in which they are listed in the menu definition.
4. Enter the syntax to identify the new item. Typically, this syntax is <button command="command name" />. (The syntax for the button element is described in "button" on page 333.)

For example, if a toolbar menu definition has three buttons, **cut**, **copy** and **paste**, and you want to add a **find** button following **paste**, move to the line following **paste** and add the find button command, as illustrated below (red indicates text that you insert).

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="true">
  <caption localeRef="btnMainCap">Edit</caption>
  <button command="cmdcut"/>
  <button command="cmdcopy"/>
  <button command="cmdpaste"/>
  <button command="cmdfind"/>
</menu>
```

A list of standard commands is provided in "List of Standard Commands" on page 199. To learn how to create a custom command that can be added to the toolbar menu, see "Custom Commands" on page 215.

5. Users will see the new toolbar menu arrangement the next time they sign on.

Adding a Dropdown List

To add a new dropdown list to a toolbar menu, follow these steps.

This procedure uses an example dropdown list with three options that the user can insert into the content.

Dropdown list option	Executes this custom command	Inserts this text string
company name	jsconame	Widgets, Inc.
address	jscoaddress	1 Main Street, New York, New York
telephone number	jstcotelnum	1-800-111-2222

1. Go to the features > external section of the configuration data.
2. On a new line, enter <command name="commandname" style="list">. Replace *commandname* with a unique name for the list. For this example, enter companyinfo.

3. Enter `<selections name="selectionlist">`. Replace *selection list* with a unique name for the selection list. For this example, enter `myselectionlist`.
4. For each item on the dropdown list that executes a command, enter `<listchoice command="command name">caption</listchoice>`. For example, to add to a dropdown list a custom command that inserts the company name, enter
`<listchoice command="jsconame">company name</listchoice>`

See Also: "Creating a List Item that Generates No Command" on page 249

NOTE

To learn how to create a custom command, see "Custom Commands" on page 215.

5. Enter closing `</selections>` and `</command>` tags. Here is a full example of the list.

```
<command name="companyinfo" style="list">
<tooltiptext>Insert company info</tooltiptext>
<selections name="myselectionlist">
    <listchoice command="jsconame">company name</listchoice>
    <listchoice command="jscoaddress">company address</listchoice>
    <listchoice command="jstcotelnum">company tel number</listchoice>
</selections>
</command>
```

6. Find the interface section of the XML configuration file.

```
<?xml version="1.0"?>
<config product="eWebEditPro">
    <!-- Valid positive values are: yes, true, 1 -->
    <!-- Valid negative values are: no, false, 0 -->
    <interface>
```

7. Within the `interface` section, move to the toolbar menu that you want to modify. (See "Finding a Toolbar Menu's Internal Name" on page 232.)
8. Move to the line within the menu tags where you want to add the new item. Buttons and dropdown lists appear on a toolbar in the sequence in which they are listed in the menu definition.
9. Enter the syntax to identify the new item. To continue with the above example, you would enter `<button command="companyinfo"/>`. (The syntax for the button element is described in "button" on page 333.)

For example, if a toolbar menu definition has three buttons, **cut**, **copy** and **paste**, and you want to add the dropdown list after **paste**, move to the line following **paste** and add the new button command, as illustrated below (red indicates text that you insert).

```
<menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="true">
    <caption localeRef="btnMainCap">Edit</caption>
    <button command="cmdcut"/>
    <button command="cmdcopy"/>
    <button command="cmdpaste"/>
    <button command="companyinfo"/>
</menu>
```

10. Place the following JavaScript on the page that displays the editor between the body's script tags. You can enter the function above or below the line that invokes the editor.

NOTE

This section does not explain how to create custom commands. See "Custom Commands" on page 215 for that procedure.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData)
{
  if ("jsconame" == strCmdName)
  {
    eWebEditPro.instances[sEditorName].editor.pasteHTML("Widgets, Inc");
  }
  else if ("jscoaddress" == strCmdName)
  {
    eWebEditPro.instances[sEditorName].editor.pasteHTML("1 Main Street, New York, New York");
  }
  else if ("jstcotelnum" == strCmdName)
  {
    eWebEditPro.instances[sEditorName].editor.pasteHTML("1-800-111-2222");
  }
}
```

Users will see the new toolbar menu arrangement the next time they sign on.

Removing a Toolbar Button or Dropdown List

NOTE

This is only element for which you cannot set the `enabled` property to "false" so that the editor will ignore its values.

1. Within the interface section of the configuration data, move to the definition of the toolbar menu that you want to modify.
2. Move to the item that you want to remove.
3. To permanently remove the button, select the entire line and press <Delete>. To temporarily remove the button, surround it with the characters that your XML editor uses to "comment out" text that is not executable code.

Users will see the new toolbar menu arrangement the next time they sign on.

Rearranging Buttons/Dropdown Lists on a Toolbar Menu

Buttons and dropdown lists appear on a toolbar menu in the sequence in which they are entered into the configuration data. For example, the following toolbar

menu definition would create this menu .

```
<menu name="editbar" . . . . . .
  <button command="cmdcut"/>
  <button command="cmdcopy"/>
  <button command="cmdpaste"/>
  <button command="cmdfind"/>
  <bar/>
</menu>
```

NOTE

The above illustration shows default images assigned to the commands in the example toolbar menu definition. However, you can modify these images using the image attribute of the command element.

To rearrange the toolbar buttons on a toolbar menu, follow these steps.

1. Within the interface section of the configuration data, move to the definition of the toolbar menu that you want to modify.
2. Move to the item that you want to move.
3. Select the entire line and cut it.
4. Move to the line where you want the item to appear and paste the text you cut in Step 3.

Users will see the new toolbar menu arrangement the next time they sign on.

Adding a Space Between Two Toolbar Menu Items

You can add a space command to separate two toolbar menu items.



Buttons with a space command

(For details, see “space” on page 353.)

To add a space command, follow these steps.

1. Within the interface section of the configuration data, move to the definition of the toolbar menu that you want to modify.
2. Move to the item after which you want to insert the space and enter <space/>.

Users will see the new toolbar menu arrangement the next time they sign on.

Adding a Separator Bar Between Two Toolbar Menu Items

Use the bar command to place a

- vertical bar on a toolbar or
- horizontal bar on a [popup menu](#)

To add a bar, follow these steps.

1. Within the interface section of the configuration data, move to the definition of the menu that you want to modify.
2. Move to the item after which you want to insert the space and enter <bar/>.

Users will see the new menu arrangement the next time they sign on.

Changing the Image that Appears on a Toolbar Button

Use the command element's image attribute to specify the image that appears on a button.

Each standard command has a default image. You can replace the default image with another standard image or a custom image. If you are creating a custom command, there is no default image. In this case, you can assign a standard or custom image to it.

If you do not assign an image to a command, the command's caption text appears on the toolbar button or menu.

A list of standard images appears in “[Images Supplied by eWebEditPro+XML](#)” on [page 360](#). If you want to create your own image, see “[Creating Your Own Images](#)” on [page 369](#).

To modify the image that appears on a button, follow these steps.

1. Within the interface section of the configuration data, move to the definition of the toolbar menu that you want to modify. (See “[Finding a Toolbar Menu’s Internal Name](#)” on [page 232](#).)
2. On that menu, find the button whose image you want to change.
3. Identify the command assigned to the button. For example, in the following example, the command assigned to the first button on the toolbar menu named editbar is cmdcut.

```
<menu name="editbar" . . . . . .
  <button command="cmdcut" />
  <button command="cmdcopy" />
  <button command="cmdpaste" />
  <button command="cmdfind" />
  <bar/>
</menu>
```

4. Move to the features section of the configuration data.

5. Find the command that you identified in Step 3.

```
<command name="cmdcut" enabled="true">
  <image key="Cut" />
  <caption localeRef="cmdCut">Cut</caption>
  <toolTipText localeRef="cmdCut">Cut</toolTipText>
</command>
```

6. Replace the command's image element with the new image.

As examples

- to specify the standard image “world”, use this code:

```
<command name="cmdcut" enabled="true">
  <image key="world" />
  <caption localeRef="cmdCut">Cut</caption>
  <toolTipText localeRef="cmdCut">Cut</toolTipText>
</command>
```

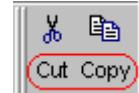
- to specify a custom image file named customcut, located in folder to which **eWebEditPro+XML** is installed, use this code:

```
<command name="cmdcut" enabled="true">
  <image src="[eWebEditProPath]/customcut.gif"/>
  <caption localeRef="cmdCut">Cut</caption>
```

```
<toolTipText localeRef="cmdCut">Cut</toolTipText>
</command>
```

7. Users will see the new image the next time they sign on.

Displaying Button Caption Text



Caption text appears on toolbar buttons in the user interface.

NOTE

To determine the alignment of text within a button, edit the `textAlignment` attribute of the toolbar menu element. See “Defining the Alignment of Caption Text” on page 243.

To display caption text for all buttons on a toolbar menu, follow these steps.

1. Within the features section of the configuration data, move to the definition of the command whose caption text you want to display.

2. Set the `visible` attribute of the `command` element definition to “`true`”. (The default value of this attribute is “`false`”.) For example:

```
<command name="cmdcut" style="icon" visible="true">
  <image key="Cut" />
  <caption localeRef="cmdCut">Cut</caption>
```

3. Move to the interface section of the configuration data.

4. Set the `showButtonsCaptions` attribute of the toolbar menu element to “`true`”.

```
<menu name="editbar" showButtonsCaptions="true" ...>
```

Users will see the caption text the next time they sign on.

To remove the display of caption text for a toolbar menu, reverse Step 4 above.

Defining the Alignment of Caption Text

You can set the alignment of button caption text using the `textAlignment` attribute of the `menu` element. The possible alignment choices are listed below.

- top
- left
- right
- bottom
- center

Button with Left Aligned Caption Text



You should only apply an `alignment` attribute to a button that displays caption text. By default, buttons do not display caption text. The procedure for displaying button caption text is described in “[Displaying Button Caption Text](#)” on page 243.

To change the alignment of button caption text, follow these steps.

1. Within the interface section of the configuration data, move to the definition of the toolbar menu whose button text alignment you want to change.
2. Edit a value for the `textalignment` attribute of the `menu` element definition. (The default value is “**top**”.)

Possible values are:

- top
- left
- right
- bottom
- center

Users will see the new alignment of the caption text the next time they sign on.

Translating Button Captions and Tool Tips

As explained in “[Modifying the Language of eWebEditPro+XML](#)” on page 265, you can translate the language of **eWebEditPro+XML**’s user interface into several foreign languages. You can translate these elements of the interface.

- button caption text (if being displayed)
- tooltip text
- items on a pull-down menu
- items on a dropdown list

For any command in the configuration data, the `ref` or `localeRef` attributes assign a code that maps to a translation value in the localization file. For example, for the cut command, the standard `ref` value is `cmdCut`.

If a localization file is assigned to the `this.locale` element in the `ewebeditprodefaults.js` file, then **eWebEditPro+XML** performs these actions before displaying the toolbar. For each command, it

1. Finds the code assigned to a command’s `ref` or `localeRef` attribute.
2. Finds the corresponding value for the code in the localization file.
3. Displays the localization file value in the interface.

Example

For example, if you assign the German localization file in the `ewebeditprodefaults.js` file, and you are displaying button caption text, when **eWebEditPro+XML** launches, it

1. Finds the `localeRef` value for the cut button, `CmdCut`.
2. Finds the translation for that text in the German localization file, `<ts id="cmdCut">Ausschneiden</ts>`.
3. Displays the text from the German localization file on the cut button, in this case, `Ausschneiden`.

To change the German text that appears, edit the localization file, in this case, `locale0407b.xml`.

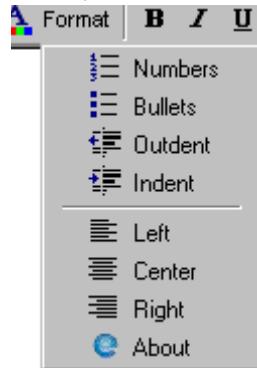
"[Modifying the Language of eWebEditPro+XML](#)" on page 265 provides more details about modifying the language of **eWebEditPro+XML**'s user interface.

To learn about creating a custom command, see "[" on page 196](#)

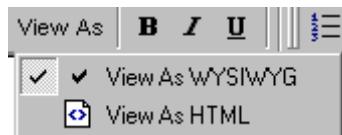
Creating a Popup Menu

You can assign a popup menu to a toolbar button. You might want to do this to provide access to many toolbar commands while limiting the toolbar to one row.

When a popup menu is assigned to a toolbar button, and a user clicks the button, the menu appears below the button, as illustrated below.



Follow these steps to create a popup menu. In this example, the menu appears on the toolbar with the text **View As**. When users click the **View As** button, they see a menu with two choices, as illustrated below.



1. Within the `features -> external` section of the configuration data, create a command to display the popup. Note that the command's caption will appear on the toolbar, indicating the menu's purpose. (For information about the syntax of the `command` element, see "[" on page 336](#).)

```
<features
  <external>
    <command name="viewasselections" style="icon" visible="true" ref="btnTxtVA" >
    </command>
```

2. Move to the interface section of the configuration data and create a popup menu. Add to the menu the buttons that you want to make available.

In this example, you are adding one button for **ViewAsWYSIWIG** and another one for **ViewAsHTML**. (For information about the syntax of the **popup** element, see “[popup](#)” on page 351.)

```
<popup name="ViewAsPopup" localeRef="btnMyViewAs">View As</caption>
<button command="cmdviewaswysiwyg" />
<button command="cmdviewashtml" />
</popup>
```

3. Add to an existing or new menu a button that invokes the command you created in Step 1.

```
<menu name="ViewAsBar" newRow="false" showButtonsCaptions="false" style="false" >
  <caption localeRef="btnViewAs">View As...</caption>
  <button command="viewasselections" />
</menu>
```

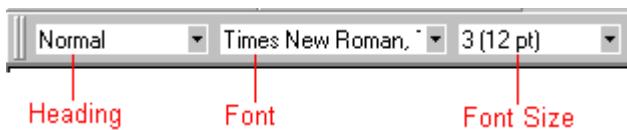
4. Within the button definition, specify a **popup** attribute. Enter the name of the menu you created in Step 3 as the value of the **popup** attribute.

```
<menu name="ViewAsBar" newRow="false" showButtonsCaptions="0" style="0" >
  <caption visible="0" localeRef="btnViewAs">View As...</caption>
  <button command="viewasselections" popup="ViewAsPopup" />
</menu>
```

Users will see the popup menu the next time they sign on.

Determining which Fonts, Font Sizes, and Headings are Available

When you install **eWebEditPro+XML**, the default toolbar provides the following fonts, font sizes and headings in selection boxes illustrated here.



Display item	Command name	Default Choices
Heading	cmdheaderlevel	<ul style="list-style-type: none"> ● Normal ● Heading 1 ● Heading 2 ● Heading 3 ● Heading 4 ● Heading 5 ● Heading 6
Font	cmdfontname	<ul style="list-style-type: none"> ● Arial, Helvetica ● Comic Sans MS ● Courier New, Courier ● Symbol ● Times New Roman, Times ● Verdana, Helvetica
Font size	cmdfontsize	<ul style="list-style-type: none"> ● 1 (8 pt) ● 2 (10 pt) ● 3 (12 pt) ● 4 (14 pt) ● 5 (18 pt) ● 6 (24 pt) ● 7 (36 pt)

Changing Available Fonts

To change the list of fonts available to users, follow these steps.

1. Within the features section of the configuration data, go to the line that begins <selections name="fontnamelist".
2. Delete or add fonts to the list.

NOTE

- A font only appears on a Web page if it is installed on the computer of the user viewing the page. You can do this with any font, including Asian fonts, but the client must be able to support the character set and that font must be on the system.
- The toolbar cannot display HTML, so if you add any fonts, such as a Japanese font, you need to type in the actual Japanese characters and not use entity names or character references. When you do this, remember to change the encoding from iso-8859-1 to your encoding type, such as, shift_jis.
- If the system does not support the character set, broken characters appear.
- If more than one font is entered for a selection, the browser tries to display the first font. If it cannot find that font, it tries to use the second one, etc. If the system has no fonts, it uses the browser's default.

Users will see the new font list the next time they sign on.

Changing Available Font Sizes

To change the list of fonts available to users, follow these steps. Note that the list cannot have more than seven sizes.

1. Within the features section of the configuration data, go to the line that begins `<selections name="fontsizelist">`.
2. Delete or add font sizes to the list. For example, to restrict users so that they can only apply font sizes 1, 3, 6 and 7, delete the lines for fonts 2, 4 and 5 (crossed out below).

```

<listchoice command="cmdfontsize1" localref="mnufS1">1 (8 pt)</listchoice>
<listchoice command="cmdfontsize2" localref="mnufS2">2 (10 pt)</listchoice>
<listchoice command="cmdfontsize3" localref="mnufS3">3 (12 pt)</listchoice>
<listchoice command="cmdfontsize4" localref="mnufS4">4 (14 pt)</listchoice>
<listchoice command="cmdfontsize5" localref="mnufS5">5 (18 pt)</listchoice>
<listchoice command="cmdfontsize6" localref="mnufS6">6 (24 pt)</listchoice>
<listchoice command="cmdfontsize7" localref="mnufS7">7 (36 pt)</listchoice>

```

Users will see the updated list of font sizes the next time they sign on.

Changing Available Headings

To change the list of headings available to users, follow these steps.

Note that the destination browser translates heading sizes into specific font sizes.

1. Within the features section of the configuration data, go to the line that begins `<selections name="headinglist">`.
2. Delete or add heading levels to the list.

For example, to restrict users so that they can only apply headings 1, 3 and 6, delete the lines for headings 2, 4 and 5 (crossed out below).

```

<selections name="headinglist" enabled="true" sorted="true">
<listchoice command="cmdheadingstd" localref="hdgtxtnorm">Normal</listchoice>
<listchoice command="cmdheading1" localref="hdgtxlvl1">Heading 1</listchoice>
<listchoice command="cmdheading2" localref="hdgtxlvl2">Heading 2</listchoice>
<listchoice command="cmdheading3" localref="hdgtxlvl3">Heading 3</listchoice>
<listchoice command="cmdheading4" localref="hdgtxlvl4">Heading 4</listchoice>
<listchoice command="cmdheading5" localref="hdgtxlvl5">Heading 5</listchoice>

```

```
<listchoice command="cmdheading6" localeref="hdgtxtlvl6">Heading 6</listchoice>
</selections>
```

Users will see the new heading list the next time they sign on.

Creating a List Item that Generates No Command

On the **eWebEditPro+XML** toolbar, you may want to create a dropdown list whose first item describes the list, or is a default value. If the user selects the first item, no action should occur. Here is an illustration of such a dropdown list.



You assign commands to list items in the configuration data. If a command is not assigned to an item, the command assigned to the list is sent. This section explains how to disable activity for a list selection.

Every command assigned to a list item is sent in the `onexeccommand` event. These commands do not need to be defined, as they do when listed on toolbars or popup menus.

Since the commands do not need to be defined, the values can be anything. If a command is defined that is not handled by the editor or by scripting, the command is ignored.

For example, you want to create a title for a list of colors, such as **Select Color**. To do this, assign a command that is not handled to the list item, for example, `noop`. You would define the list like this.

```
<selections name="fontcolorlist" enabled="true" sorted="true">
  <listchoice command="noop">Select Color</listchoice>
  <listchoice command="cmdfontcolor">Color palette</listchoice>
  <listchoice data="#FF0000">red</listchoice>
  <listchoice data="#0000FF">blue</listchoice>
  <listchoice data="#00FF00">green</listchoice>
</selections>
```

If a user selects **Select Color** from the dropdown list, nothing happens.

Dynamically Changing the Editor

This section explains how to dynamically change the default options for each of **eWebEditPro+XML**'s features using JavaScript

- on the server side, when the editor is created
- on the client side, after the editor is created

You can use these two approaches together if you wish.

Dynamically Creating Configuration Data on the Server Side

The server can dynamically create configuration data when the editor requests it. The data can be populated from a database. You can do this because the configuration of XML data is not limited to a flat file. This is one of XML's most powerful characteristics.

To use this method, set the editor's config parameter to a URL that returns configuration data as XML. The parameter must be set *before* the **eWebEditPro+XML** editor is created. Once the editor is created, the toolbar is static unless you also implement the technique described in “[Dynamically Changing the Editor on the Client Using JavaScript](#)” on page 251.

In the sample code below, the server (in this example, running ASP) returns XML data as it would appear in a flat config.xml file. But, the server is creating the file dynamically.

You use a URL parameter (**id=1**, in this example) to provide necessary information to the server. URL parameters are optional.

```
<script language="JavaScript1.2">
eWebEditPro.parameters.reset(); // set all parameters to their default values
// Request the configuration.
eWebEditPro.parameters.config = eWebEditPro.parameters.path + "config.asp?id=1";
</script>
```

Avoiding Problems When Dynamically Changing the Toolbar on the Server

If you plan to generate configuration data for the toolbar on the server, keep the following points in mind.

- If you are doing this with ColdFusion, you should turn off debug information. If you do not, you will receive a confusing error message.
You can turn off debug information site wide in the Cold Fusion Administrator Panel, under Debugging. Or, you can do this for a single page using the following code <CFSETTING SHOWDEBUGOUTPUT="NO" >.
- Caching may make it difficult to view differences in the editor. You should eliminate caching on the browser and also in the code.
For example, at the top of an ASP page, the following code forces the browser to flush the cache <RESPONSE.EXPIRES=0>.
- If you dynamically create the toolbar in Netscape, the editor cannot access cookie information.

Dynamically Changing the Editor on the Client Using JavaScript

You can dynamically alter the toolbar after the editor is created using client-side JavaScript. You can add or remove buttons, dropdown lists, etc. Typically, the toolbar is changed in the editor's onready event as a result of user interaction.

Two sections of this documentation provide the information about client-side scripting.

- “[Disabling and Enabling Menu Items within Scripting](#)” on page 251 provides tips and techniques for using client scripting to access menu functionality.
- “[The Toolbar Object Interface](#)” on page 258 provides the API definition of the Toolbar object contained within the **eWebEditPro+XML** interface. The Toolbar object interface contains properties and methods that let you control menu, button, and command functionality.

Disabling and Enabling Menu Items within Scripting

You can use client scripting to access menu functionality. This includes menu creation, command creation, and display status. This section describes how to disable and enable a menu item through scripting.

Accessing Menus and Commands

To access menus, use the Toolbars method of the **eWebEditPro+XML** control. This method returns a reference to the menu control object.

```
var objInstance = ewebEditPro.instances[seditorname];
var objMenu = objInstance.editor.Toolbars();
```

To access a command, use the following method in the toolbar object.

Method: CommandItem(CommandName As String) As CCommandItem

See Also: “[Method: CommandItem](#)” on page 65

The following are methods in the Command object used to affect enable status.

Method: setProperty(Name As String, Value As Variant)

See Also: “[Method: setProperty](#)” on page 124

Method: getPropertyBoolean(Name As String) As Boolean

See Also: “[Method: getPropertyBoolean](#)” on page 91

Property: CmdGray As Boolean

See Also: “[Property: CmdGray](#)” on page 134

Enabling and Disabling a Command

To enable or disable a command, first retrieve the interface to menus using the Toolbar method:

```
var objInstance = eWebEditPro.instances[seditorname];
var objMenu = objInstance.editor.Toolbars();
```

Then, use the Toolbar object to gain access to a specific command.

```
var objCommand = objMenu.CommandItem("cmdcut");
```

To disable an icon command, set the CmdGray property to true.

```
objCommand.setProperty("CmdGray", true);
```

To enable an icon command, set the CmdGray property to false.

```
objCommand.setProperty("CmdGray", false);
```

Example

Below are functions that disable and enable a command item.

```
function DisableCommand(seditorname, scommandname)
{
var objInstance = eWebEditPro.instances[seditorname];
var objMenu = objInstance.editor.Toolbars();
objMenu.CommandItem(scommandname).setProperty("CmdGray", true);
}

function EnableCommand(seditorname, scommandname)
{
var objInstance = eWebEditPro.instances[seditorname];
var objMenu = objInstance.editor.Toolbars();
objMenu.CommandItem(scommandname).setProperty("CmdGray", false);
}
```

Customizing the Popup Button

This section explains how to customize a popup button and window. To see an example of a popup button and the window containing **eWebEditPro+XML** that appears when the user presses the button, follow this path (beginning with the Windows Start button).

Start > Programs > Ektron > eWebEditPro 4 > Samples > Multiple Editors

Scroll to the bottom of the window to see the Popup button ()¹, whose default caption is "Edit".

There are two approaches to customizing the popup button and window. You can

- Change values in `ewebeditprodefaults.js` and `ewebeditpromessages.js`. All properties within these files that start with `popup` affect the popup button or window. For example, the popup button caption is declared as `popupButtonCaption` in `ewebeditpromessages.js`.
See Also: ["The `ewebeditprodefaults` File" on page 291](#); ["The `ewebeditpromessages` File" on page 292](#)
- Change the properties using JavaScript. Two objects, `buttonTag` and `popup`, are part of the `parameters` object. For example,
`eWebEditPro.parameters.buttonTag.value = "Edit Description";`
`eWebEditPro.createButton("btnDesc", "Desc");`

NOTE

You *cannot* use JavaScript at run time to change popup properties in `ewebeditpromessages.js` or `ewebeditprodefaults.js`.

The JavaScript objects correspond as shown below.

Parameter	<code>ewebeditprodefaults.js</code>
<code>buttonTag.start</code>	<code>popupButtonTagStart</code>
<code>buttonTag.value</code>	<code>popupButtonCaption</code> (in <code>ewebeditpromessages.js</code>)
<code>buttonTag.end</code>	<code>popupButtonTagEnd</code>
<code>popup.url</code> <i>See Also:</i> "Property: <code>popup</code>" on page 171	<code>popupurl</code>
<code>popup.windowName</code>	<code>popupWindowName</code>
<code>popup.windowFeatures</code>	<code>popupWindowFeatures</code>

Parameter	ewebeditprodefaults.js
popup.query	popupQuery
styleSheet See Also: "Style Sheets" on page 430	styleSheet

"Customizable JavaScript Files" on page 291 explains how to edit the JavaScript files. The rest of this section explains how to customize the popup button using JavaScript.

Customizing the createButton Command

By default, when you create a popup edit button for **eWebEditPro+XML** (using the `<input type=button>` element), a standard HTML button with a caption of **Edit** is created.

In JavaScript, use the `createButton` method to create the button.

```
ewebeditPro.createButton("btnName", "contentFieldName");
```

To customize the popup button, use the parameters `buttonTag` object. You can set it in `ewebeditprodefaults.js` or in JavaScript before calling the `createButton` method.

You specify the popup button caption in `ewebeditpromessages.js` as `popupButtonCaption`. You can also set it in JavaScript using the `value` property.

The following are values for the `type` property (or `popupButtonTagType` in `ewebeditprodefaults.js`). They let you determine the form of the popup edit button.

Value	Description	HTML
inputbutton	Standard Input Button	<code><input type="button"></code>
button	Button	<code><button>Edit</button></code>
image	Graphic Image (.gif or .jpg)	<code></code>
imagelink	Hyperlinked Image	<code><a></code>
hyperlink	Hyperlinked text	<code><a>Edit</code>
custom	Custom HTML	(custom)

For example:

```
<textarea name=Summary rows=10 cols=70>
<script language="JavaScript1.2">
```

```
eWebEditPro.parameters.reset();
eWebEditPro.parameters.buttonTag.type = "imagelink";
eWebEditPro.createButton("btn1", "Summary");
</script>
```

You can assign custom attributes using the tagAttributes property (or popupButtonTagTagAttributes in ewebeditprodefaults.js). For example:

```
eWebEditPro.parameters.buttonTag.tagAttributes = "onmouseover='mymouseover()'" ;
```

If an image type is selected, you can customize the image using the imageTag object. Set the imageTag properties to specify the attributes of the tag. For example:

```
eWebEditPro.parameters.buttonTag.imageTag.src = "myimage.gif";
eWebEditPro.parameters.buttonTag.imageTag.alt = "Click to edit";
eWebEditPro.parameters.buttonTag.imageTag.width = 40;
eWebEditPro.parameters.buttonTag.imageTag.height = 20;
```

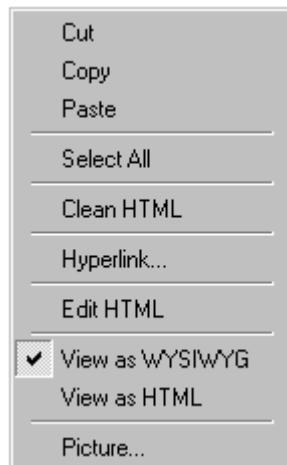
To create your own custom HTML, use the start and end properties (popupButtonTagStart and popupButtonTagEnd in ewebeditprodefaults.js). The string 'eWebEditPro.edit("the-element-name")' will be inserted between start and end.

For example:

```
eWebEditPro.parameters.buttonTag.start = "<object ...><param name='editjs' value=''";
eWebEditPro.parameters.buttonTag.end = "'>...</object>" ;
```

Customizing Context Menus

This section describes how to customize context-sensitive menus that appear when a user right-clicks the mouse. For example, the following menu appears if text has been selected when the user right-clicks the mouse within the editor.



A different context menu appears if no text was selected or if the cursor is within a table.

The next sections explain how to [remove](#) individual commands from the context menus, and how to [suppress](#) all of them. You cannot [add](#) items to a context menu.

Removing Commands from a Context Menu

To remove a command from the context menu, set its `enabled` attribute to “`false`”. Note that this change also removes the command from the toolbar menu. You cannot remove a command from the context menu and leave it on the toolbar menu.

For example, to remove the copy command from the context menu, edit the configuration data as illustrated below. (To learn more about editing configuration data, see “[Editing the Configuration Data](#)” on page 312.)

```
<standard>
  <command name="cmdCopy" enabled="false">
    </command>
</standard>
```

Context Menu Commands and their Internal Names

In order to remove a command from a context menu, you must know its internal name. This table lists the internal name of most commands.

Menu Command	Internal Name
Cut	cmdcut
Copy	cmdcopy
Paste	cmdpaste
Select all	<i>cannot be removed</i>
Clean HTML	cmdclean
Hyperlink...	cmdhyperlink
Edit HTML	edithtml
Insert HTML	edithtml
View as WYSIWYG	cmdviewaswysiwyg
View as HTML	cmdviewashtml
Picture...	cmdmfumedia

Suppressing the Context Menu

The configuration data's interface element has a `context` attribute that determines whether context menus appear when a user right clicks the mouse.

```
<config product="eWebEditPro">
<interface name="standard" visible ="true" allowCustomize="false" context="false"

```

If the context attribute is set to “**true**”, the context menu appears when the right mouse is clicked. If the attribute is set to “**false**,” the context menu is suppressed.

The default value for context is “**true**”.

The Toolbar Object Interface

This section describes the API definition of the Toolbar object contained within the **eWebEditPro+XML** interface. It assumes that you have moderate expertise in JavaScript, HTML, and ActiveX technology.

The Toolbar object interface contains properties and methods that let you control menu, button, and command functionality. See “[Toolbars Object](#)” on page 19.

To retrieve the Toolbar object, use the **eWebEditPro+XML** Toolbar method.

```
var objMenu = objEditor.Toolbars();
```

This section explains the following topics.

- [Defining Menus and Commands](#)
- [Toolbar Object Quick Reference](#)
- [Command Object Quick Reference](#)
- [Script Example](#)
- [Command Values](#)

Defining Menus and Commands

The menus and commands are typically defined in the config.xml file assigned to the editor. This file defines the command names, how they look, and where they appear in the user interface.

You can add scripting to modify or supplement the XML data. The server side can anticipate changes required for a user and dynamically generate a customized XML from database information. To accomplish this, it may be necessary to interact with commands at a level below user interaction. This low level interaction can include creating a toolbar using commands supported by a script and removing commands not supported by a script.

The following sections contain information to help a developer perform these operations. In doing so, it provides a reference to the API.

Toolbar Object Quick Reference

See “[Toolbars Object](#)” on page 19.

Command Object Quick Reference

See “[Command Item Object](#)” on page 25.

Script Example

The following code is a sample JavaScript 1.2 function that creates a menu, creates buttons and commands, and grays out command items.

```
function AffectMenusAndCommands(seditorname)
{
    var objInstance = eWebEditPro.instances[seditorname];
    var objMenu = objInstance.editor.Toolbars();

    // This shows how to create a menu.
    objMenu.ToolbarAdd("SampleMenu", "This is an External Menu", 0, 0, 0, 0, "");

    // This shows how to add commands to a menu.
    objMenu.CommandAdd("jssample1", "Sample Command 1", "Sample Command 1", "key", 0, 0, "SampleMenu", 0, -1);
    objMenu.CommandAdd("jssample2", "Sample Command 2", "Sample Command 2", "key", 0, 0, "SampleMenu", 0, -1);
    objMenu.CommandAdd("jssample3", "Sample Command 3", "Sample Command 3", "key", 0, 0, "SampleMenu", 0, -1);
    objMenu.CommandAdd("jssample4", "Sample Command 4", "Sample Command 4", "key", 0, 0, "SampleMenu", 0, -1);
    objMenu.CommandAdd("jssample5", "Sample Command 5", "Sample Command 5", "key", 0, 0, "SampleMenu", 0, -1);

    // This shows how to disable an existing command.
    objMenu.CommandItem("cmdcut").setProperty("CmdGray", true);
    objMenu.CommandItem("cmdpaste").setProperty("CmdGray", true);
}
```

Command Values

The following command values are collections of API parameter value types.

- [etbToolbarOptions](#)
- [etbToolbarStyles](#)
- [etbCaptionAlignment](#)
- [etbToolbarLocation](#)
- [etbToolbarModifications](#)
- [etbCommandOptions](#)
- [etbCommandStyles](#)
- [etbCommandModifications](#)
- [etbErrorValues](#)

etbToolbarOptions

The menu creation options are assembled using the bitwise OR operation.

Value	Description
0x1	Invisible

0x2	New menu row
0x4	ShowCaption
0x8	Wrap rather than scroll
0x10	Show tool tips
0x20	Show customize selection menu button
0x40	Floating

etbToolbarStyles

Toolbar Styles. Used when creating a toolbar.

Value	Description
0	Icon Bar
1	Pulldown Menu
2	Tab List
3	Popup or context menu

etbCaptionAlignment

Caption alignment of buttons. Use these values when creating a toolbar or command.

The effects are seen only when a toolbar is floating or a command is defined to show its caption.

See Also: "Defining the Alignment of Caption Text" on page 243

Value	Align caption to the
0	No alignment of caption
1	Top
2	Bottom
3	Left

4	Right
5	Center

etbToolbarLocation

Toolbar Location. Used when creating a toolbar.

Value	Align toolbar to the
0	No alignment of toolbar. Default (top) is used.
1	Top
2	Bottom
3	Left
4	Right
5	Form

etbToolbarModifications

Toolbar modifications allowed. Used when modifying a toolbar.

Value	Description
0	Delete command wherever it is.
1	Remove from toolbar but keep on customization list.
2	Move from the selection to the toolbar.
3	Set as pressed in or checked.
4	Set as un-pressed or unchecked.
5	Toolbars only - clear all commands.

6	Disable the item.
7	Enable the item.

etbCommandOptions

Command creation options are bitwise or'ed together. Used when creating or modifying a command.

Value	Description
0x1	Invisible
0x2	Initially shows as disabled

etbCommandStyles

Command Styles. Used when creating a command.

Value	Description
0	Use the Default button style when the function it represents has no dependence on other functions. For example, a Save File operation can be performed at any time. Further, when the button is depressed, it springs back again when the function is finished.
1	The Check style should be used when the function it represents is a toggle of some kind. For example, when using a RichTextBox control, selected text can be either bold or not. Thus, when the button is depressed, it stays depressed until it is pressed again.
2	Not supported.
3	The separator style has no function except to create a button that is eight pixels wide. Use the separator style to create a button that separates one button from another. <i>See Also:</i> " Adding a Separator Bar Between Two Toolbar Menu Items " on page 241 Or, use it to enclose the group of buttons with the ButtonGroup style.
4	The placeholder style functions as a "dummy" button. Use this button to create a space on the Toolbar control where you want to have another control appear (such as a ComboBox or ListBox control).

5	Dropdown list box.
6	Text edit box.

etbCommandModifications

Command modifications allowed. Used when modifying a command.

Value	Description
0	Delete all instances of a command.
1	Command is removed from the toolbar but kept on the customization list.
2	Display command on toolbar or menu.
3	Set as pressed in or checked.
4	Set as un-pressed or unchecked.
5	For toolbars only; clear all commands.
6	Disable the item.
7	Enable the item.

etbErrorValues

Error definitions. These are returned by the Menus interface methods.

Value	Description
0	No error.
1	Functionality not supported with this version.
2	No toolbars have been created for any operation.

3	Invalid ID given for a command or toolbar.
4	Unknown location requested for text or command.
5	Internal error.
6	The specified toolbar does not exist.
7	Error using definition file.
8	Definition not found.
9	Configuration can't be used, even if given.
10	Error creating item.

Modifying the Language of eWebEditPro+XML

You can modify the language in which **eWebEditPro+XML**'s dialog boxes, menus and messages appear, as well as the language of the content. You can even spell check the content in a foreign language.

You might want to modify the language because your users speak a different language, or because you want to customize the standard text provided with the system. This chapter explains how to accomplish these goals.

NOTE *Messages, strings and labels intended for developers are not translated.*

See Also: “Translating Button Captions and Tool Tips” on page 244

This section explains the following topics.

- [How eWebEditPro+XML Determines the Language of the User Interface](#)
- [Locale Files](#)
- [Translating eWebEditPro+XML's User Interface](#)
- [Languages Supported by Windows](#)
- [Working with non-English Content](#)
- [Using the Languages Sample](#)
- [Displaying Menus and Dialogs in a non-European Language](#)
- [Setting the Language of Spell Checking](#)
- [Modifying Standard Text \(including English\)](#)

How eWebEditPro+XML Determines the User Interface Language

When a user launches a page that hosts **eWebEditPro+XML**, it

1. checks the value assigned to the `this.locale` variable in `ewebeditprodefaults.js`.
 - If the `this.locale` parameter is the default value (`this.path + " "`), **eWebEditPro+XML** displays dialogs, menus, and messages in the language selected in Windows' regional settings.
 - If the `this.locale` parameter is set to a specific locale file, **eWebEditPro+XML** uses the strings in the corresponding file. For example, `this.locale = this.path + "locale040ab.xml"` instructs **eWebEditPro+XML** to use the `locale040ab.xml` locale file, which displays the user interface in Spanish.
 - If **eWebEditPro+XML** cannot find the locale file specified, it displays text in the language selected in Windows' regional settings.

2. displays the translation string for the text elements of the user interface.
- For example, the **eWebEditPro+XML** configuration data assigns a translation code to each text string. In the configuration data, the Cut button's translation code is cmdCut. If the `this.locale` is set to `locale040ab.xml` (Spanish), when the editor displays the tooltip text for the Cut button, **eWebEditPro+XML** finds `cmdCut` within `locale040ab.xml` and displays the Spanish translation string: "Cortar."

Locale Files

eWebEditPro+XML's locale files translate menus, dialog boxes, and messages to a foreign language. "[Standard Locale Files](#)" on page 266 lists the languages into which **eWebEditPro+XML** is translated, and the locale code of each file.

NOTE The locale ActiveX control property may contain XML content, but it typically refers to a directory or a locale file.

The locale files are installed by default to the directory to which you install **eWebEditPro+XML**. The file name consists of the following elements:

`localelanguage identifier.xml`

The four-character language identifier specifies the language and country. For example, `locale0407b.xml` = German.

Also within each locale file, an `xml:lang` attribute specifies the language code. For example, `xml:lang="de"` for German.

Standard Locale Files

Language(Country)	Locale file	Language Code
Default (English)	<code>locale0000b.xml</code>	
Arabic	<code>locale0401b.xml</code>	<code>ar</code>
Danish	<code>locale0406b.xml</code>	<code>da</code>
Dutch	<code>locale0413b.xml</code>	<code>nl</code>
English	<code>locale0409b.xml</code>	<code>en</code>
French	<code>locale040cb.xml</code>	<code>fr</code>
German	<code>locale0407b.xml</code>	<code>de</code>
Hebrew	<code>locale040db.xml</code>	<code>he</code>

Language(Country)	Locale file	Language Code
Italian	locale0410b.xml	it
Japanese	locale0411b.xml	ja
Korean	locale0412b.xml	ko
Portuguese (standard)	locale0816b.xml	pt
Russian	locale0419b.xml	ru
Simplified Chinese (China (PRC))	locale0804b.xml	zh-cn
Simplified Chinese (Hong Kong)	locale0c04b.xml	zh-hk
Simplified Chinese (Macau)	locale1404b.xml	zh-mo
Simplified Chinese (Singapore)	locale1004b.xml	zh-sg
Swedish	locale041db.xml	sv
Spanish	locale040ab.xml	es
Traditional Chinese (Taiwan)	locale0404b.xml	zh-tw

Translating eWebEditPro+XML's User Interface

To have eWebEditPro+XML appear in this language	Do this
English	Nothing -- the interface automatically appears in that language
One of the translated languages	See " Displaying the User Interface in a Translated Language " on page 268.
Not a translated language but on the list of Windows-supported languages	Translate text in several files into that language. See " Translating the User Interface to a Windows-Supported Language " on page 269.

To have eWebEditPro+XML appear in this language	Do this
Not one of the Windows-supported languages	You cannot display eWebEditPro+XML user interface menus and dialogs in non-Windows supported language

Displaying the User Interface in a Translated Language

IMPORTANT! This procedure is only required if the client computer's Regional Setting language is different from the language in which you want to display eWebEditPro+XML.

To select which translated language to use, follow these steps.

1. Navigate to the directory to which you installed eWebEditPro+XML.
2. Open ewebeditprodefaults.js.
3. Find the line that begins `this.locale`.
4. Between the quotes following `this.path+`, insert the locale file of the translation language. For example, to display eWebEditPro+XML in Spanish, change that line so that it looks like this:
`this.locale = this.path + "locale040ab.xml";`

For a list of languages and corresponding locale identifiers, see "Standard Locale Files" on page 266.

5. Save ewebeditprodefaults.js.

From now on, eWebEditPro+XML references the locale file specified in ewebeditprodefaults.js for the text on tooltips, menus, dialogs, etc. It will also display system messages from files whose name includes the two-character alphabetical code for the specified country. To continue the example of translating into Spanish, eWebEditPro+XML would reference these files (`es` is the country code for Spanish):

- `installnowes.htm`
 - `introes.htm`
 - `ewebeditpromessageses.js`
 - `section508tabletextes.js`
6. Update the ewebeditpro.js file according to "Updating ewebeditpro.js" on page 277.
 7. Update the ewebeditprodefault.js file according to "Updating ewebeditprodefault.js" on page 278.
 8. Update the eweutil.js file according to "Updating eweutil.js" on page 279.

See Also: "Translating Button Captions and Tool Tips" on page 244

Translating the User Interface to a Windows-Supported Language

To translate eWebEditPro+XML's dialogs boxes, menus and messages into one of the approximately 100 languages supported by Windows, translate the English in the following files into the new language. Instructions for translating each file appear below.

See Also: "Languages Supported by Windows" on page 279

IMPORTANT!

If the non-English language is listed on "Standard Locale Files" on page 266, the files are already translated. See "Displaying the User Interface in a Translated Language" on page 268.

Description of file	File name	Read this section
The locale file	locale0000b.xml	"Translating the Locale File (localexxxxb.xml)" on page 270
System message and status bar text	ewebeditpromessages.js	"Translating the Messages File (ewebeditpromessages.js)" on page 275
The automatic client installation Web page	installnow.htm	"Translating the Automatic Client Installation Web Page (installnow.htm)" on page 271
HTML page that prompts user to install eWebEditPro+XML	intro.htm	"Translating the Page that Prompts User to Install eWebEditPro+XML (intro.htm)" on page 273
The Section 508 table dialog	section508tabletext.js	"Translating the Section 508 Tables Dialog" on page 276
The eWebEditPro JavaScript file	ewebeditpro.js	"Updating ewebeditpro.js" on page 277
The eWebEditPro defaults file	ewebeditprodefaults.js	"Updating ewebeditprodefaults.js" on page 278
The eWebEditPro utility file	eweputil.js	"Updating eweputil.js" on page 279

Translating the Locale File (`localexxxxb.xml`)

1. Browse to the eWebEditPro folder.
2. Open locale0000b.xml in a text editor.
3. Change the `xml:lang` value to the new language's two character country code. To find the country code, see "Languages Supported by Windows" on page 279.
4. For example, change

```
<locale version="2" product="eWebEditPro" xml:lang="en">
```

 to

```
<locale version="2" product="eWebEditPro" xml:lang="bg">
```
5. Replace the English text between each set of `<ts>` tags with the translation text. For example, replace
`<ts id="abt">Cut</ts>`
 with
`<ts id="abt">Bulgarian term for cut</ts>`

Notes regarding values

- One item on each menu must have a unique underlined character. The user presses this character to access the menu option using a keyboard instead of a mouse.
 Use the ampersand (&) to underline a character. For example, enter &Help to display Help.
- You can underline a character inside nested menus, as in **Edit -> Copy and Tool -> Customize**.
- In Asian translations, use a Roman letter in parentheses at the end of word and underline it. For example, (H).
- Do not underline descending characters such as g, j, y, p, and q.
- If using a plain text editor instead of an XML editor, use the entity name instead of the character for the characters listed below.

Character	Entity
&	&
<	<
>	>
Line break	

6. Save the file under a new name. To create the new name, use the pattern `localexxxxb.xml`, where xxxx represents the last four characters of the corresponding language's identifier. To obtain the language identifier, see "Languages Supported by Windows" on page 279.

For example, if the language is Bulgarian, the file's name would be `locale0402b.xml`.

Translating the Automatic Client Installation Web Page (`installnow.htm`)

1. Within the `ewebeditpro5` folder, open the `clientinstall` folder.
2. Open `installnow.htm`.
3. Within the `<head>` tags, look for the following line:
`<meta http-equiv=Content-Type content="text/html; charset=iso-8859-1">`
- If the `<meta>` tag is missing, add it.
4. If necessary, replace the `charset` value for the language (for example, `iso-8859-1`). See "Languages Supported by Windows" on page 279 for the character set identifiers.
5. Translate the text. Text that requires translation is shown below in red. The actual file may vary from what is shown. This sample was taken from **eWebEditPro+XML** version 2.1.

WARNING!

Do not use the contents shown below. Start with the `installnow.htm` file provided with **eWebEditPro+XML**.

```
<!-- Revision Date: 2001-08-22 -->
<html>
<head>
<meta http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<title>eWebEditPro Installation</title>
<script language="JavaScript1.2" src="../ewebeditpro.js"></script>
<style>
P { font-size : small; font-family : verdana, helvetica; }
H1 { font-family : verdana, helvetica; }
H2 { font-family : verdana, helvetica; }
A { font-family : verdana, helvetica; }
BODY { font-size : small; }
</style>

<script language="JavaScript1.2">
<!--
function reloadOpener()
{
if (top.opener && !top.opener.closed)
{
top.opener.location.reload();
}
//-->
</script>

</head>
```

```

<body onunload="reloadOpener()">

<p align="center"><h2 align="center">
eWebEditPro <br>Automatic <br>Download and Installation
</h2></p>

<form method="post">
<input type=hidden name="DoneMsg" value="&lt;p&gt; &lt;/p&gt;&lt;p align=center&gt;&lt;font
face='Arial' size=4&gt;Installation complete.&lt;/font&gt;&lt;/p&gt;">
<input type=hidden name="RestartMsg" value="&lt;p&gt; &lt;/p&gt;&lt;p align=center&gt;&lt;font
face='Arial' size=4&gt;Please restart Windows to complete the installation.&lt;/font&gt;&lt;/p
&gt;">

<p align="center"><font size=-1>
<br>
If successful, the words "Installation complete" will appear in the box below.
</font></p>
<script language="JavaScript1.2">
<!--
eWebEditPro.onready = onReadyHandler;
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE;
eWebEditPro.parameters.installPopup = null;
eWebEditPro.create("DoneMsg", "100%", 200);

function onReadyHandler()
{
  document.loadingMsg.style.visibility = "hidden";
  eWebEditPro.refreshStatus();
  if (eWebEditPro.autoInstallExpected())
  {
    eWebEditPro.instances[0].load(document.forms[0].elements.RestartMsg.value);
    document.body.onunload = ""; // don't reload the opener window
  }
}
//-->
</script>
<p><font size=-1>
<script language="JavaScript1.2">
<!--
document.write('If a small red <font color="red"><b>X</b></font>appears, try downloading the
');
document.write('<a href="' + eWebEditProDefaults.clientInstall + '">');
document.write('client installation program</a>and running it. ');
// -->
</script>
</font></p>
<p><font size=-1>
For additional assistance, visit <a href="http://www.ektron.com/support"
target=_blank>Ektron's support page</a>.
</font></p>
<p align="center">
<input type="button" name="btnClose" value="Close" onclick="self.close()">
</p>
</form>

```

```
</body>
</html>
```

6. Save the file as installnowxx.htm, where xx is the two letter country code. The country codes are listed in "Languages Supported by Windows" on page 279. For example, installnowbg.htm for Bulgarian.

Translating the Page that Prompts User to Install eWebEditPro+XML (intro.htm)

1. Within the ewebeditpro5 folder, open the clientinstall folder.
2. Open intro.htm.
3. Within the <head> tags, look for the following line:
`<meta http-equiv=Content-Type content="text/html; charset=iso-8859-1">`
If the <meta> tag is missing, add it.
4. If necessary, replace the charset value for the language (for example, iso-8859-1). See "Languages Supported by Windows" on page 279 for the character set identifiers.
5. Translate the text. Text that requires translation is shown below in red. The actual file may vary from what is shown. This sample was taken from eWebEditPro+XML version 2.1.

WARNING!	Do not use the contents shown below. Start with the intro.htm file provided with eWebEditPro+XML.
-----------------	---

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- Copyright 2001 Ektron, Inc. -->
<!-- Revision Date: 2001-05-16 -->
<html>
<head>
<meta http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<title>Installing eWebEditPro</title>
</head>

<style>
P { font-size : small; font-family : verdana, helvetica; }
H1 { font-family : verdana, helvetica; }
H2 { font-family : verdana, helvetica; }
A { font-family : verdana, helvetica; }
BODY { font-size : small; }
</style>

<body>

<blockquote><blockquote>
<p align="center"><h2 align="center">
eWebEditPro <br>Automatic <br>Download and Installation
</h2></p>
<p>The page you are trying to view contains Ektron's eWebEditPro editor. It will appear within your browser. It allows you to enter content for web pages as easily as using a word processor.</p>
<p>Before you can use eWebEditPro, it must be downloaded into your browser. When you click the <b>Install Now</b>button at the bottom of this page, eWebEditPro will be automatically downloaded and installed. This process may take several minutes depending on the speed of your
```

network connection. Once downloaded, eWebEditPro will <i>not need to download again</i> unless upgrading to a newer version. </p>

<p>You must have authorization to install programs on your computer. </p>

<p>Pictures similar to the one shown below may appear. The first is from Ektron. The others, if they appear, are from Microsoft. If they do appear, please click Yes to proceed.</p>

<p align="center">

</p>

<p>Check your Internet Explorer security setting. It must be set at Medium or lower to permit downloading and running ActiveX controls. From the Tools menu, select Internet Options and click the Security tab. </p>

<p align="center">

</p>

<p>Depending on your security settings, you may also be prompted with the questions shown here. Click Yes each time.</p>

<blockquote>

<p>Do you want to allow software such as ActiveX controls and plug-ins to run?</p>

<p>A script is accessing some software (an ActiveX control) on this page which has been marked safe for scripting.
Do you want to allow this?</p>

</blockquote>

<p>Click on Install Now to proceed, or Cancel if you do not want to install eWebEditPro.</p>

</blockquote></blockquote>

<p align="center">

<form>

<input type=button value="Install Now "onclick="location.href='installnow.htm'">

<input type=button value="Cancel" onclick="self.close()">

</form>

More information on Ektron's eWebEditPro

</p>

</body>

</html>

6. Change the name of the installation file listed within intro.htm to the file you saved under a new name at the end of "Translating the Automatic Client Installation Web Page (installnow.htm)" on page 271. To continue with our example, you would change

```
<input type=button value="Install Now "onclick="location.href='installnow.htm'">
to
<input type=button value="Install Now "onclick="location.href='installnowbg.htm'">
```

7. To display the Internet Explorer Security Options dialog in the selected language, follow these steps.

- Save an image of the dialog as a GIF file named as "ieoptionsxx.gif", where xx is the two letter language code used earlier.

- Reduce the image to 75% of the original size using a commercially available image editor.
- Save the image as "ieoptions2xx.gif", where xx is the two letter language code used earlier.
- Copy the image to the clientinstall directory.
- In the intro.htm file, change the IMG tag as shown below, where xx is the language code, WWW is the width of the image in pixels, and HHH is the height.

```

```

8. Save the file as introxx.htm, where xx is the two letter country code. The country codes are listed in ["Languages Supported by Windows" on page 279](#). For example, introbg.htm for Bulgarian.

Translating the Messages File (ewebeditpromessages.js)

1. Within the ewebeditpro5 folder, open ewebeditpromessages.js.
2. Translate the text. Text that requires translation is shown below in red. The actual file may vary from what is shown. This sample was taken from eWebEditPro+XML version 2.1.

Because this JavaScript file can be included on a web page with any charset encoding, (for example, iso-8859-1, utf-8, big5), the translated text must be ASCII, which is compatible with all encodings. The Web page encoding uses the charset defined in a meta tag, for example, UTF-8.

```
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
```

Escaping Special and Unicode Characters to ASCII

- Special characters whose code is hex 80 to hex FF must be escaped using \xNN, where NN is the hex value. For example, 'suçon' would be 'su\xe7on' because the 'ç' character is code U+00E7.
- Unicode characters whose code is above hex FF must be escaped using \uNNNN, where NNNN is the hex value. For example, the string with three Unicode characters with codes U+65E5 U+672C U+8A9E would become '\u65e5\u672c\u8a9e'.

TIP Ektron provides a Web page that escapes the characters so you do not need to convert them by hand. See the Test Languages Web Page for details. It allows you to translate the messages strings in a native, human-readable encoding, view the escaped strings in the Web page, and copy them into your translated messages file.

WARNING!	Do not use the contents shown below. Start with the ewebeditpromessages.js file provided with eWebEditPro.
-----------------	--

```
// Copyright 2000, Ektron, Inc.  
// Revision Date: 2001-01-30
```

```

/* Modify this file to set your preferred messages in the language of your choice. */

var eWebEditProMessages =
{
    popupButtonCaption: "Edit"
, installPrompt: "Click OK to install eWebEditPro."
, waitingToLoad: "Waiting to load"
, loading: "Loading"
, doneLoading: "Done loading"
, errorLoading: "Error loading"
, saving: "Saving"
, doneSaving: "Done saving"
, querySave: "Click OK to preserve changes when moving to another page.\nClick Cancel to discard changes."
, confirmAway: "Any changes will be lost."
, saveFailed: "Unable to save. Continue and lose content?"
, sizeExceeded: "Content is too large to save. Please reduce the size and try again."
, clientInstallMessage: '<br><font face="Arial" size=1 color=red>eWebEditPro is not installed. Click to <A href="' + eWebEditProDefaults.clientInstall + '">install eWebEditPro</A>.</font>'
, elementNotFoundMessage: '<br><font face="Arial" size=2 color=red><b>Unable to find content field (typically a hidden field) within a form.</b><br>Please check the following:<ul><li>Form tag is required<li>Content field is required and must match the name specified when creating the editor<li>Content field must be declared prior to creating the editor</ul>Name specified:</font>'
, invalidFormMethodMessage: '<br><font face="Arial" size=2 color=red><b>The form method must be set to "post".</b> For example, &lt;form method="post"&gt;. The submit will fail using "get".</font>'
}

```

NOTE

The message strings for elementNotFoundMessage and invalidFormMethodMessage do not need to be translated. These messages are intended for developers, not end-users.

3. Save the file as ewebeditpromessagesxx.js, where xx is the two letter country code. The country codes are listed in ["Languages Supported by Windows" on page 279](#). For example, ewebeditpromessagesbg.js for Bulgarian.

Translating the Section 508 Tables Dialog

1. Within the ewebeditpro5 folder, open section508tabletext.js.
 2. Translate the text. Text that requires translation is shown below in red. The actual file may vary from what is shown. This sample was taken from eWebEditPro+XML version 4.0.0.12.
- Because this JavaScript file can be included on a Web page with any charset encoding, (for example, iso-8859-1, utf-8, big5), the translated text must be ASCII, which is compatible with all encodings. The Web page encoding uses the charset in a meta tag, for example, UTF-8.

```
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
```

See Also: ["Escaping Special and Unicode Characters to ASCII" on page 275](#)

WARNING! Do not use the contents shown below. Start with the section508tabletext.js file provided with eWebEditPro.

```
// Copyright 2000-2003, Ektron, Inc.
// Revision Date: 2002-Mar-06

/* Modify this file to set your preferred messages in the language of your choice. */

var Section508TableMsges =
{
    LabelForm: "Section 508 Table Properties"
,   LabelHeadRows: "Heading Rows:"
,   LabelHeadCols: "Heading Columns:"
,   LabelSummary: "Summary:"
,   LabelCaption: "Caption:"
,   LabelHCaption: "Horizontal Caption Alignment:"
,   OptionNotSet: "Not Set"
,   OptionLeft: "Left"
,   OptionCenter: "Center"
,   OptionRight: "Right"
,   LabelVCaptionAlign: "Vertical Caption Alignment:"
,   OptionTop: "Top"
,   OptionBottom: "Bottom"
,   LabelBtnDone: "OK"
,   LabelBtnCancel: "Cancel"
,   MsgIllegalHeadRows: "The number of Heading Rows is greater than the maximum allowed."
,   MsgInvalidHeadRows: "An integer is expected as the number of Heading Rows."
,   MsgIllegalHeadCols: "The number of Heading Columns is greater than the maximum allowed."
,   MsgInvalidHeadCols: "An integer is expected as the number of Heading Columns."
,   MsgNoEditorforSave: "The Editor is not available. Section 508 Properties are not saved."
,   MsgNoEditorforLoad: "The Editor is not available. Section 508 Properties cannot be
retrieved. Please try again."
,   MsgWarnHeadCols: "No Heading Columns can be set."
,   LabelMax: "Max."
}
```

3. Save the file as section508tabletext`xx`.js, where xx is the two letter country code. The country codes are listed in ["Languages Supported by Windows" on page 279](#). For example, section508tabletextbg.js for Bulgarian.

Updating ewebeditpro.js

This procedure adds the new language to the list of supported languages.

1. Browse to the ewebeditpro folder.
2. Open the ewebeditpro.js file in a text editor.
3. If needed, add the country code of the new language to the list of languages, which occurs in the section of the file copied below. Separate each code with a comma. Do not include spaces.

The list does not need to be in alphabetical order.

IMPORTANT!

If the language contains a suffix for the country code, add it to the first list with "zh-tw". If the language is just a language code, add it to the second list with "da,de" etc.

```
var strTranslatedLangCodes = "zh-tw";
if (strTranslatedLangCodes.indexOf(strLanguageCode) == -1)
{
    strLanguageCode = strLanguageCode.substring(0, 2);
    var strTranslatedLanguages = "da,de,es,fr,it,ja,ko,nl,pt,zh";
    if (strTranslatedLanguages.indexOf(strLanguageCode) == -1)
```

4. Add the two character language code between the parentheses at the end of this line:

```
eWebEditProMsgsFilename = defaultMsgsFilename(); =
defaultMsgsFilename();
```

For example, if the language is Spanish, you would add es:

```
eWebEditProMsgsFilename = defaultMsgsFilename(es);
```

5. Save the file. Do not rename it.

Updating ewebeditprodefaults.js

This specifies which installnowxx.htm file to open to automatically install **eWebEditPro+XML**.

1. Browse to the ewebeditpro folder.
2. Open the ewebeditprodefaults.js file in a text editor.
3. If needed, add the country code of the new language to the list of languages, which occurs in the section of the file copied below. Separate each code with a comma. Do not include spaces.
The list does not need to be in alphabetical order.

IMPORTANT!

If the language contains a suffix for the country code, add it to the first list with "zh-tw". If the language is just a language code, add it to the second list with "da,de" etc.

```
:var strTranslatedLangCodes = "zh-tw";
if (strTranslatedLangCodes.indexOf(strLanguageCode) == -1)
{
    strLanguageCode = strLanguageCode.substring(0, 2);
    var strTranslatedLanguages = "da,de,es,fr,it,ja,ko,nl,pt,zh";
    if (strTranslatedLanguages.indexOf(strLanguageCode) == -1)
```

4. Add the two character language code (as shown in red) to the following line of code.

```
this.installPopupUrl = this.path + "clientinstall/" +
defaultInstallFilename("es"); // parameters.installPopup.url
```

5. Save the file. Do not rename it.

Updating eweputil.js

1. Browse to the `ewebeditpro` folder.
2. Open the `eweputil.js` file in a text editor.
3. Find the line that includes `return strLanguageCode;`:
4. Replace `strLanguageCode` with the new language's two character country code. To find the country code, see "[Languages Supported by Windows](#)" on [page 279](#). For example, if the language is Spanish, the line would look like this:
`return es;`
5. Save the file. Do not rename it.

Languages Supported by Windows

Terms on the Supported Languages Table

Identifier

An identifier is a hexadecimal value that specifies a language and country. The identifier's four characters appear in the name of each locale file. See *Also: "Locale Files" on page 266*

Country Code

A country code is an abbreviation for a language. Some language codes include a two-letter suffix that specifies a country.

Character Set

This is the preferred character set for each language's encoding.

Language	Identifier (hex)	Language Code	Charset
Language Neutral	0x0000		
Afrikaans	0x0436	af	
Albanian	0x041c	sq	Ü
Arabic	0x0401	ar	iso-8859-6 or windows-1256
Azeri (Latin)	0x042c		

Azeri (Cyrillic)	0x082c		
Basque	0x042d	eu	
Belarussian	0x0423	be	iso-8859-5 or windows-1251
Bulgarian	0x0402	bg	iso-8859-5 or windows-1251
Burmese	0x0455		
Catalan	0x0403	ca	
Chinese (Taiwan)	0x0404	zh-tw	big5
Chinese (PRC)	0x0804	zh-cn	gb2312
Chinese (Hong Kong SAR, PRC)	0x0c04	zh-hk	
Chinese (Singapore)	0x1004	zh-sg	
Chinese (Macau SAR)	0x1404		
Croatian	0x041a	hr	iso-8859-2 or windows-1250
Czech	0x0405	cs	iso-8859-2 or windows-1250
Danish	0x0406	da	iso-8859-1 or windows-1252
Dutch (Netherlands)	0x0413	nl	iso-8859-1 or windows-1252
Dutch (Belgium)	0x0813	nl-be	
English (United States)	0x0409	en-us	iso-8859-1 or windows-1252
English (United Kingdom)	0x0809	en-gb	
English (Australian)	0x0c09	en-au	

English (Canadian)	0x1009	en-ca	
English (New Zealand)	0x1409	en-nz	
English (Ireland)	0x1809	en-ie	
Estonian	0x0425	et	iso-8859-4 or windows-1257
Faeroese	0x0438	fo	
Farsi	0x0429	fa	
Finnish	0x040b	fi	iso-8859-1 or windows-1252
French (Standard)	0x040c	fr	iso-8859-1 or windows-1252
French (Belgian)	0x080c	fr-be	
French (Canadian)	0x0c0c	fr-ca	
French (Switzerland)	0x100c	fr-ch	
French (Luxembourg)	0x140c	fr-lu	
German (Standard)	0x0407	de	iso-8859-1 or windows-1252
German (Switzerland)	0x0807	de-ch	
German (Austria)	0x0c07	de-at	
German (Luxembourg)	0x1007	de-lu	
German (Liechtenstein)	0x1407	de-li	
Greek	0x0408	el	iso-8859-7 or windows-1253
Hebrew	0x040d	he	iso-8859-8 or windows-1255

Windows 2000: Hindi. This is Unicode only.	0x0439	hi	
Hungarian	0x040e	hu	iso-8859-2 or windows-1250
Icelandic	0x040f	is	
Indonesian	0x0421	in	
Italian (Standard)	0x0410	it	iso-8859-1 or windows-1252
Italian (Switzerland)	0x0810	it-ch	
Japanese	0x0411	ja	shift_jis (An alternative is iso-2022-jp or euc-jp)
Korean	0x0412	ko	euc-kr (An alternative is iso-2022-kr)
Korean (Johab)	0x0812	ko	
Latvian	0x0426	lv	iso-8859-4 or windows-1257
Lithuanian	0x0427	lt	iso-8859-4 or windows-1257
Macedonian	0x042f	mk	iso-8859-5 or windows-1251
Malay (Malaysian)	0x043e	ms	
Norwegian	0x0414	no	iso-8859-1 or windows-1252
Polish	0x0415	pl	iso-8859-2 or windows-1250
Portuguese (Brazil)	0x0416	pt-br	
Portuguese (Standard)	0x0816	pt	iso-8859-1 or windows-1252

Romanian	0x0418	ro	iso-8859-2 or windows-1250
Russian	0x0419	ru	iso-8859-5 or windows-1251
Serbian (Cyrillic)	0x0c1a	sr	iso-8859-5 or windows-1251
Serbian (Latin)	0x081a	sr	
Slovak	0x041b	sk	iso-8859-2 or windows-1250
Slovenian	0x0424	sl	iso-8859-2 or windows-1250
Spanish (Traditional Sort)	0x040a	es	iso-8859-1 or windows-1252
Spanish (Mexican)	0x080a	es-mx	
Spanish (Modern Sort)	0x0c0a	es	
Spanish (Guatemala)	0x100a	es-gt	
Spanish (Costa Rica)	0x140a	es-cr	
Spanish (Panama)	0x180a	es-pa	
Spanish (Dominican Republic)	0x1c0a	es-do	
Spanish (Venezuela)	0x200a	es-ve	
Spanish (Colombia)	0x240a	es-co	
Spanish (Peru)	0x280a	es-pe	
Spanish (Argentina)	0x2c0a	es-ar	
Spanish (Ecuador)	0x300a	es-ec	
Spanish (Chile)	0x340a	es-cl	
Spanish (Uruguay)	0x380a	es-uy	

Spanish (Paraguay)	0x3c0a	es-py	
Spanish (Bolivia)	0x400a	es-bo	
Spanish (El Salvador)	0x440a	es-sv	
Spanish (Honduras)	0x480a	es-hn	
Spanish (Nicaragua)	0x4c0a	es-ni	
Spanish (Puerto Rico)	0x500a	es-pr	
Sutu	0x0430	sx Sutu	
Swahili (Kenya)	0x0441		
Swedish	0x041d	sv	iso-8859-1 or windows-1252
Swedish (Finland)	0x081d	sv-fi	
Thai	0x041e	th	iso-8859-11 or windows-874
Turkish	0x041f	tr	iso-8859-9 or windows-1254
Ukrainian	0x0422	uk	iso-8859-5 or windows-1251
Urdu (Pakistan)	0x0420	ur	
Urdu (India)	0x0820		
Uzbek (Latin)	0x0443		
Uzbek (Cyrillic)	0x0843		
Vietnamese	0x042a	vi	windows-1258

Working with non-English Content

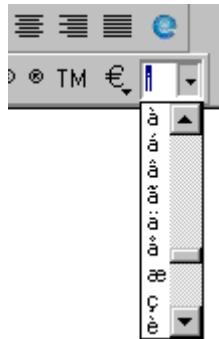
IMPORTANT!

Editor content can be in any language supported by the browser, even if the system does not support applications in that language.

eWebEditPro+XML content can be in any language supported by the browser, which is largely controlled by Windows regional settings. The content can contain multiple languages simultaneously. The HTML languages sample (<http://localhost/ewebeditpro5/samples/html/languages/languages.htm>) demonstrates a page with several languages.

Accented Characters

For accented characters, the user can select from the list of special and extended characters available on the standard toolbar (see below) or type from the keyboard.



Also, non-English keyboards usually have the characters printed.

You can enter special characters with an English keyboard (using Alt keys) using the IME (input editor). See **Windows Control Panel > Regional Options > Input Locales**.

Using the Languages Sample

The following eWebEditPro+XML sample screen illustrates how the locale file affects the editor.

Start > Programs > Ektron>ewebeditpro5 > Samples > HTML Samples > Languages

Select the language of your choice. Then, move the cursor over any toolbar button and notice that the tooltip appears in the selected language. Also, if you click an icon that displays a dialog box (such as Insert Picture), the dialog box appears in the selected language.

Displaying Menus and Dialogs in a non-European Language

If you specify an Asian or Middle Eastern language locale for eWebEditPro+XML, and you are running a European (for example, English) version of Windows, the non-European characters may appear as question marks (?) instead of the ideogram.

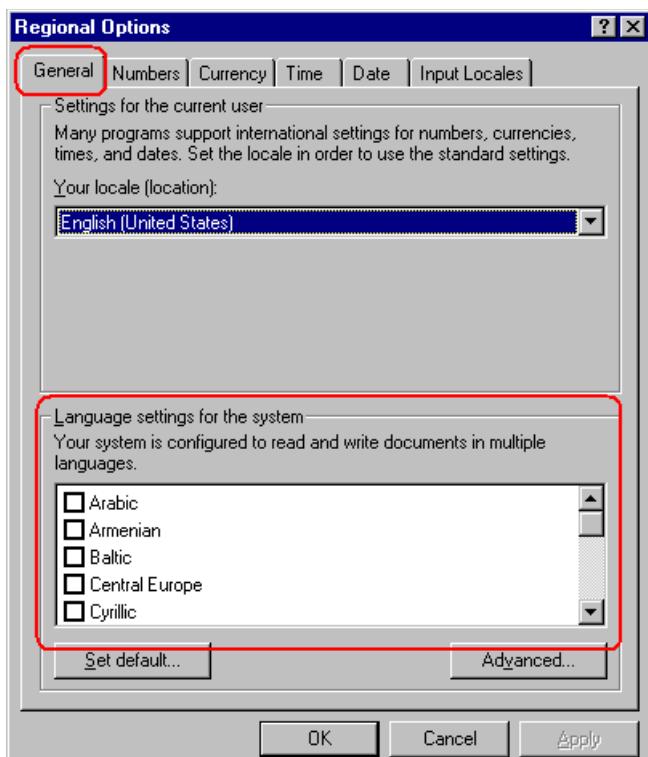
Similarly, accented Latin (that is, European) characters may appear without their accent marks on an Asian version of Windows.

These problems occur because the language's character set is not loaded in the client PC's Windows operating system. Any Unicode character that does not have a corresponding character in the character set (that is, code page table) appears as a question mark (?).

How to Fix in Windows XP and 2000

To display **eWebEditPro+XML** menus and dialogs in a non-European language on a European version of Windows 2000 or XP, set the system default language to the language you wish to display. To do this for Windows 2000, follow these steps.

1. Open the Control Panel.
2. Open Regional Options.
3. Click the **General** tab. The lower half of the dialog displays the languages currently configured for your system.

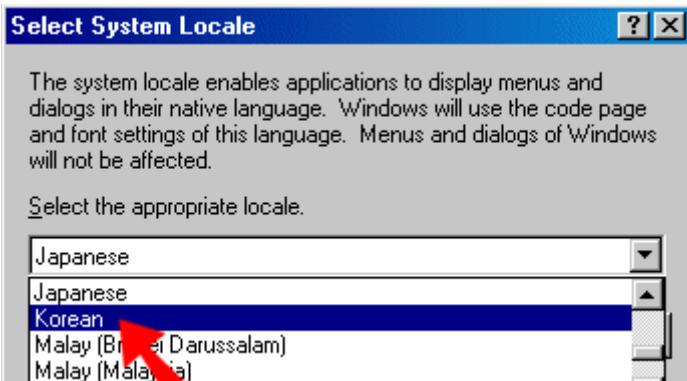


4. Check the language you want to use. You may be prompted for the Windows CD-ROM to install language files.

NOTE

Your **locale**, in the top half of the dialog, does not affect the default language.

5. Click the **Set default** button. The "Select System Locale" dialog appears.



6. Select a language from the list. For example, Korean, as shown here.
7. Click **OK** to close the dialog boxes. You are probably prompted to restart Windows.

How to Fix in Windows NT, Me, 98, 95

Older versions of Windows do not support all languages. To display a language, you need a version of Windows localized for that language. For instance, Japanese Windows is required to view **eWebEditPro+XML** menus and dialogs in Japanese.

NOTE

The content in the editor does not have this limitation, only the menus and dialogs. The languages supported in the editor content are only limited by the browser.

Setting the Language of Spell Checking

There are three ways that you can specify which language to use when checking spelling.

- Set the language in Microsoft Word, version 2000 or later. To do this, open Word and select **Tools >Language > Set Language**.
- Set the language in the configuration XML data using the langid attribute of the `<spellcheck>` element. For more information, see "["langid" on page 405](#)".
- Specify the language in JavaScript by passing `IData` parameter in the `ExecCommand` method. (For more information, see "["Creating a Custom Command" on page 215](#).)

Modifying Standard Text (including English)

Most of this chapter explains how to display **eWebEditPro+XML**'s standard text in a non-English language. However, you may want to modify the standard text of any language, including English, for several reasons:

- the text is an error message, and you want to provide organization-specific directions
- you want to brand the product with your corporate ID
- you want the spelling to follow regional conventions. For example, in British English, *colour* is the correct spelling.

To modify the **eWebEditPro+XML**'s standard text, read the table below to determine the language category that you are modifying.

To modify text in this language	Read this section
American English	" Modifying American English Text " on page 289
Any other language into which eWebEditPro+XML is translated	" Modifying the Standard Text of a Translated Language " on page 289
Any other Windows-supported language	" Modifying the Standard Text of a Windows-Supported Language " on page 290

Location of Translated Strings

The user interface text resides in these files.

Type of string	Folder/File name
Most system text, including <ul style="list-style-type: none"> tooltip text menu options dialog box field labels and responses messages colors 	ewebeditpro5\locale\language identifier\b.xml
Status bar text and system messages, especially those concerning loading and saving a document	ewebeditpro5\ewebeditpromessages\language code.js
Automatic client installation Web page	ewebeditpro5\clientinstall\installnow\language code.htm

Type of string	Folder/File name
HTML page that prompts user to install eWebEditPro+XML	ewebeditpro5\clientinstall\intro\language code.htm
The Section 508 table dialog	ewebeditpro5\section508tabletext\language code.js
Dialogs that appear when user launches client.exe	Contact Ektron for help translating

Modifying American English Text

IMPORTANT! Since the default language is American English, eWebEditPro+XML displays system text in English but does not refer to the American English locale file (locale0409b.xml) unless you explicitly list it in ewebeditprodefaults.js.

To modify English system text, follow these steps.

1. Assign the American English locale file in ewebeditprodefaults.js.
 - Navigate to the ewebeditpro folder.
 - Open the ewebeditprodefaults.js file.
 - Find the `this.locale` variable.
 - Change the variable so that it refers to the American English locale file. It should look like this: `this.locale = this.path + "locale0409b.xml";`
 2. Refer to ["Location of Translated Strings" on page 288](#) to determine which file to edit.
 3. Open the appropriate file, change the text, and save the file. To edit the locale file, edit `locale0409b.xml`. To edit any other system file, edit the generic version, that is, the file without a two character country code, such as `ewebeditpromessages.js`.
- For example, to change tooltip text, open `locale0409b.xml`, find the existing text, and replace it with new text.

Modifying the Standard Text of a Translated Language

This section assumes that eWebEditPro+XML is already set to a translated language. For instructions on how to do this, see ["Displaying the User Interface in a Translated Language" on page 268](#).

1. Refer to ["Location of Translated Strings" on page 288](#) to determine which file to edit.
2. Edit the version of the file that includes the two-character code of the non-English language. For example, if the language is Spanish and you want to modify tooltip text, open `locale040ab.xml`, modify and save.

Modifying the Standard Text of a Windows-Supported Language

"[Translating the User Interface to a Windows-Supported Language](#)" on page 269 explains how to translate the text in all of the system files. To modify any of this text, follow the appropriate instructions for translating and simply change the translation string.

Customizable JavaScript Files

eWebEditPro+XML provides five JavaScript files that let you customize many attributes of the HTML element used to place the ActiveX control in a Web page. The files let you customize how the editor appears and functions on a Web page.

The customizable files are

- `ewebeditpro.js`
- `ewebeditprodefaults.js`
- `ewebeditpromessages.js`
- `ewebeditproevents.js`
- `ewebeditpromedia.js`

This section describes each file.

The `ewebeditpro.js` File

The `ewebeditpro.js` file has one element, described below.

Element	Description
<code>eWebEditPro Path</code>	Enter or edit the path to the directory to which eWebEditPro+XML is installed.

The `ewebeditprodefaults.js` File

The `ewebeditprodefaults.js` file has many attributes. Because the following attributes are also properties of the Parameters Object, they are described in the sections listed below.

- “Property: clientInstall” on page 168
- “Property: editorGetMethod” on page 182
- “Property: embedAttributes” on page 168
- “InstallPopup Object” on page 10
- “Property: maxContentSize” on page 168
- “Property: objectAttributes” on page 169
- “Event: onblur” on page 187
- “Event: ondblclickelement” on page 186
- “Event: onexeccommand” on page 186

- “Event: onfocus” on page 187
- “Property: path” on page 169
- “InstallPopup Object” on page 10

The following attributes are documented in “Active X Properties” on the pages listed below.

- “Property: baseURL” on page 150
- “Property: CharSet” on page 160
- “Property: Config” on page 160
- “Method: HideAbout” on page 93
- “Property: License” on page 162
- “Property: Locale” on page 162
- “Property: StyleSheet” on page 163
- “Property: Title” on page 163
- wddx (for compatibility with Release 1.8)

The onexeccommand attribute is described in [“The ewebeditproevents File” on page 295](#).

As described in [“Changing Parameter Values” on page 768](#), you would make changes to this file that apply to *all* occurrences of the editor. To change any of these values for a *single* occurrence of the editor, you would insert JavaScript onto the page that invokes the editor.

The ewebeditpromessages File

The attributes in the ewebeditpromessages.js file determine the text of buttons and popup messages that appear when using **eWebEditPro+XML**.

Attribute	Determines the text that appears	Default Message
popupButtonCaption	On the button that launches the popup window that contains eWebEditPro+XML .	Edit
installPrompt	In a dialog box when the client installation is needed.	Click OK to install eWebEditPro+XML
waitingToLoad	In the status bar while the editor is waiting to load.	Waiting to load
loading	In the status bar while the editor is loading.	Loading

Attribute	Determines the text that appears	Default Message
doneLoading	In the status bar when the editor finishes loading.	Done loading
errorLoading	In the status bar when the editor encounters an error and cannot load.	Error loading
saving	In the status bar when the editor is saving content.	Saving
doneSaving	In the status bar when the editor has saved content.	Done saving
querySave (used only with Internet Explorer)	In a dialog box if the user moves to another page before the content is saved. (Note: The content is saved when the user clicks the submit button.)	<p>Click OK to preserve changes when moving to another page. Click Cancel to discard changes.</p> <p>Note</p> <hr/> <p>This message only appears when using Internet Explorer.</p> <hr/> <p>See Also: "Disabling the "Click OK to Preserve Changes" Message" on page 294.</p>
confirmAway (used only with Internet Explorer)	In a dialog box if the user indicates that he/she does not want to save changes.	Any changes will be lost.
saveFailed	In a dialog box if the user clicked save but the editor cannot save the content.	Unable to save. Continue and lose content?
sizeExceeded	In a dialog box if the amount of content that user wants to save exceeds the maxContentSize.	Content is too large to save. Please reduce the size and try again.

Attribute	Determines the text that appears	Default Message
clientInstallMessage	<p>If the user opens a page with the editor using Netscape when eWebEditPro+XML is not yet installed. It also appears if the editor could not be properly initialized for some reason, such as security settings prevented the installation of the CAB files.</p> <p>The message appears below the textarea element that appears in place of the editor and prompts the user to install the client software.</p>	<pre>
 eWebEditPro+XML is not installed. Click to install eWebEditPro+XML .</pre>
elementNotFoundMessage	<p>When eWebEditPro+XML cannot find the named content element.</p> <p>The message appears below the editor.</p>	<pre>'
 Unable to find content field (typically a hidden field) within a form.
Please check the following: Form tag is required Content field is required and must match the name specified when creating the editor Content field must be declared prior to creating the editor Name specified: '</pre>
invalidFormMethodMessage	<p>When the form element's method is not set to post.</p>	<pre>'
 The form method must be set to "post". For example, &lt;form method="post"&gt;. The submit will fail using "get".'</pre>

Disabling the "Click OK to Preserve Changes" Message

If a user attempts to move to another Web page, the **Click OK to preserve changes when moving to another page. Click Cancel to discard changes** message appears.

If you want to suppress the message, add one of following JavaScript lines to a page with the editor.

```
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_SAVE
```

or

```
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE
```

The ewebeditproevents File

When the user presses a button on the toolbar or double-clicks an element (for example, a hyperlink or image) in the content, an event is raised. When the event fires, it can run a JavaScript function.

The ewebeditproevents.js file contains JavaScript event handler functions that perform actions. These actions could include inserting HTML into the content (for example, the trademark symbol) and opening a window to a hyperlink that was double-clicked.

The ewebeditproevents.js file can call the following event handler functions that you can define in a custom JavaScript file. In this way, you can customize what happens when the event fires.

Event	Determines How to Respond When
onDblClickElementHandler	A user double-clicks.
onDblClickHyperlinkHandler	A user double-clicks on a hyperlink. See Also: “Determining which Fonts, Font Sizes, and Headings are Available” on page 246
onExecCommandHandler	A toolbar button is pressed or the user selects an item from the context-sensitive menu.

To add your own commands, define one or both of the following.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData) { }
eWebEditPro.onexeccommand = your_custom_event_handler;
```

For more information, see “Event Handler Functions” on page 300.

To add your own media file handler, define:

```
function eWebEditProMediaSelection(sEditorName) { }
```

For more information, see “The ewebeditpromedia File” on page 296.

To add your own double-click element handlers, define one or more of the following:

```
function eWebEditProDblClickElement(oElement) { }
function eWebEditProDblClickHyperlink(oElement) { }
function eWebEditProDblClickImage(oElement) { }
function eWebEditProDblClickTable(oElement) { }
eWebEditPro.ondblclickelement = your_custom_event_handler;
```

For more information, see “Event Handler Functions” on page 300.

Add your custom JavaScript file to the list in ewebeditpro.js, as shown below.

```
var eWebEditProIncludes = [
  "ewebeditproevents.js",
  "ewebeditprodefaults.js",
  "ewebeditpromedia.js",
```

```
eWebEditProMsgsFilename,  
"ewep.js",  
"mycustomevents.js"];
```

The ewebeditpromedia File

This file lets you customize the external media file selection process. It is referenced during media selection to let the user insert (and possibly upload) an image into the editor.

This file contains only one function.

```
function eWebEditProMediaSelection(sEditorName)
```

In the `sEditorName` parameter, enter the name of the editor that calls the function. (See Also: “[Appendix A: Naming the eWebEditPro+XML Editor](#)” on page 778.)

This function is called when an external image selection mechanism is specified in the `transport type` property of the `mediafiles` feature in the `config.xml` data. The function determines which page to load. This page should perform the media upload plus any other custom operations, such as login or advertisements.

Below is an example entry in the configuration data.

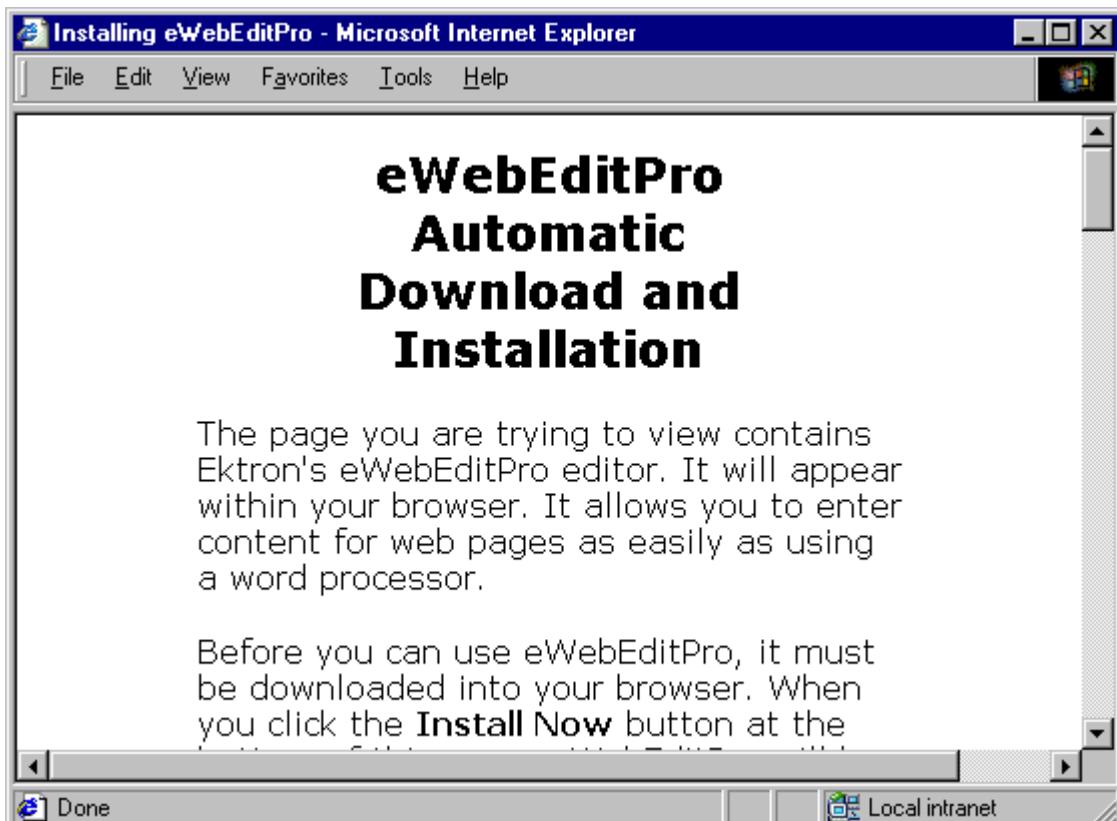
```
<mediafiles>  
<transport type="coldfusion/database/mediamanager.cfm">
```

The above entry causes the page `coldfusion/database/mediamanager.cfm` to load during the image selection process.

For more information, see “[Customizing the Popup Button](#)” on page 253.

Client Installation Pages

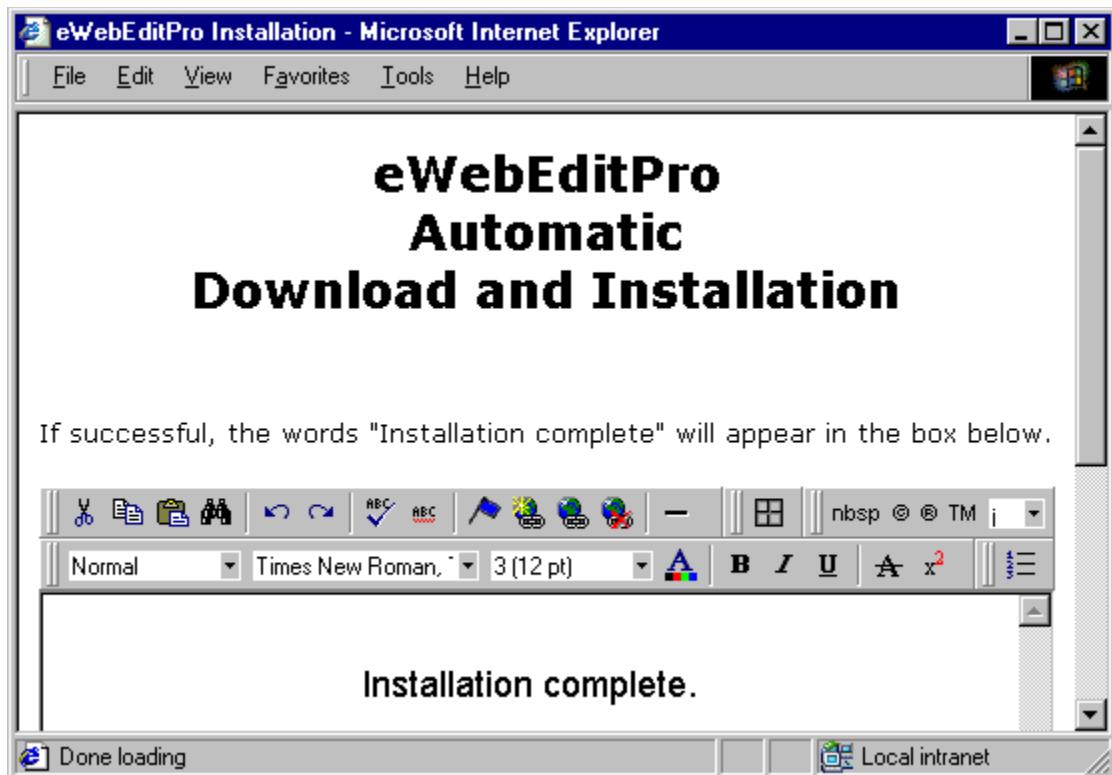
The first time a user on a client PC accesses a Web page using Internet Explorer that has a new or upgraded version of **eWebEditPro+XML**, an.htm page appears. The page provides information about the installation and prompts the user to continue with the installation or cancel.



By default, this .htm page is named intro.htm, and is installed in /ewebeditpro5/clientinstall/intro.htm.

Another .htm page, installnow.htm (installed into the same directory) is invoked from intro.htm. installnow.htm displays **Please wait while**

eWebEditPro+XML is being installed, then notifies the user whether or not the installation was successful.



Customizing the Client Installation Pages

If you want to customize these .htm pages (for example, to change the text), save the file under a different name and make changes to the copy. Otherwise, future upgrades will overwrite your changes.

If you change the intro.htm file, you also need to change the reference to the file and pathway in the ewebeditprodefaults.js file. In that file, the intro.html page is defined at the `this.installPopupUrl` property, as illustrated below.

```
function eWebEditProDefaults()
{
.
.
this.installPopupUrl = this.path + "clientinstall/intro.htm";
```

If you want to display the intro.htm page before loading a page that includes eWebEditPro+XML, you may call the `eWebEditPro.autoInstallExpected()` method to determine if the client computer would automatically install eWebEditPro+XML.

To popup a window with the intro.htm page in JavaScript, call
`eWebEditPro.installPopup.open();`

For additional JavaScript methods and properties, see “[InstallPopup Object](#)” on page 10.

Disabling the Installation Pages

If you want to disable the client installation pages, you have two choices.

- In ewebeditprodefaults.js, set `this.installPopupUrl = "";`
- In JavaScript, set `eWebEditPro.parameters.installPopup = null;`

What Happens When Auto Install Fails or is Cancelled

If the auto install is cancelled or fails, it only prompts again if one of three things happens.

- the user clicks the message **Try to automatically download and install** (This message appears below the textarea field on the same line that displays **eWebEditPro+XML** is not installed. Click to install **eWebEditPro+XML**.)
- a new version of **eWebEditPro+XML** is available on the server
- a cookie installed to suppress this message expires (by default, after 3 years)

The **Try to automatically download and install** message is defined in `ewebeditpromessages.js` (`clientAutoInstallMessage`) and, so, is customizable.

If you want to disable this feature using JavaScript, set `eWebEditPro.autoInstallCookie = null` before creating an instance of the editor.

You can change the cookie’s expiration date using JavaScript. To do this, before creating an instance of the editor, set `eWebEditPro.autoInstallCookie.expiresInSeconds = n` (where `n` is the number of seconds before expiration).

JavaScript Objects

This section describes the following topics.

- [the JavaScript object model](#)
- [the eWebEditPro+XML JavaScript object's properties, methods, and events](#)
- [event handler functions](#)
- [double-click element handlers](#)
- [the eWebEditProExecCommandHandlers array](#)
- [the toolbar reset command](#)
- [reacting to the creation of a toolbar](#)
- [the redisplay toolbars command](#)
- [the instance object](#)
- [the parameters object](#)
- [the eWebEditProUtil JavaScript Object](#)

The JavaScript Object Model

For a diagram of the JavaScript object model, see “[eWebEditPro+XML Object Model](#)” on page 2.

Examples

```
var oMedia = eWebEditPro.instances[i].editor.MediaFile();
eWebEditPro.parameters.buttonTag.value = "Edit this section";
```

JavaScript lets you add custom properties dynamically at run-time. In contrast, ActiveX control objects cannot be extended in this way.

The instance JavaScript object can be used to store data associated with a given editor on a Web page. For example,

```
eWebEditPro.instances[i].customProperty = "myvalue";
```

JavaScript Object Properties, Methods and Events

See “[eWebEditPro Object](#)” on page 4

Event Handler Functions

- ["Event: eWebEditProExecCommand" on page 191](#)
- ["Event: eWebEditProReady" on page 191](#)
- ["Event: eWebEditProMediaSelection" on page 192](#)

Double-Click Element Handlers

To add your own double-click element handler, define a JavaScript function in your Web page to run as shown below.

```
eWebEditProDblClickElement(oElement)
{
    return true or false
}
```

The eWebEditProDblClickElement function runs when certain elements are double-clicked. It may be easier, however, to define the applicable handler function for a specific object.

The hyperlink, image, and table element objects have their own functions that run when they are double-clicked.

See Also:

- "Event: eWebEditProDblClickElement" on page 193
- "Event: eWebEditProDblClickHyperlink" on page 193
- "Event: eWebEditProDblClickImage" on page 194
- "Event: eWebEditProDblClickTable" on page 194

The eWebEditProExecCommandHandlers Array

The eWebEditProExecCommandHandlers array helps you add custom commands or define command event handlers for standard commands. You can define the code to process a command in `customevents.js` or the page that displays the editor.

For example (taken from `editorwithstyle.htm`),

```
function setStyleSheet(sEditorName, strCmdName, strTextData, lData)
{
    var strStyleSheet = myStyleSheets[strCmdName];
    if ("string" == typeof strStyleSheet)
    {
        eWebEditPro.instances[sEditorName].editor.setProperty("StyleSheet", strStyleSheet);
        bStylesheetDisabled = false;
    }
}

eWebEditProExecCommandHandlers["jsstyledefault"] = setStyleSheet;
eWebEditProExecCommandHandlers["jsstyle1"] = setStyleSheet;
eWebEditProExecCommandHandlers["jsstyle2"] = setStyleSheet;
eWebEditProExecCommandHandlers["jsstyleparagraph"] = setStyleSheet;
eWebEditProExecCommandHandlers["jsstylenone"] = function(sEditorName, strCmdName,
    strTextData, lData)
{
    eWebEditPro.instances[sEditorName].editor.disableAllStyleSheets();
    bStylesheetDisabled = true;
}
eWebEditProExecCommandHandlers["jshighlight"] = function(sEditorName, strCmdName,
    strTextData, lData)
{
```

```
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdselstyle", ".highlight", 0);
}
```

Note that each array entry defines a handler for one command. If more than one command uses the same function, set each array entry to the same function (for example, `setStyleSheet`). The syntax is:

```
eWebEditProExecCommandHandlers[command_name] = your_handler_function
```

The handler must be a function with the same parameters as `eWebEditProExecCommand`, namely,

```
function(sEditorName, strCmdName, strTextData, lData)
```

ExecCommandHandlersArray Parameters

Parameter	Type	Description
sEditorName	String	<p>The name of the occurrence of eWebEditPro+XML. To access the eWebEditPro+XML methods, use <code>eWebEditPro.instances[sEditorName].editor</code>.</p> <p><i>See Also:</i> “Appendix A: Naming the eWebEditPro+XML Editor” on page 778</p> <p>Note: If your Web server is running ASP.NET, use this syntax: <code>eWebEditPro.instances[sEditorName].editor</code></p>
strCmdName	String	<p>The name of the JavaScript command that was just executed (if a standard command) or to be executed (if a custom command). Standard commands begin with <i>cmd</i>; custom commands begin with <i>js</i>.</p> <p>An example of a JavaScript function that executes after a standard command executes is checking spelling within all instances of the editor on a page. If the user presses the spellcheck button in one editor, he is prompted to continue to the next editor. If confirmed, the command is sent to the next editor on the page and will loop back to the first editor on the page.</p>
strTextData	String	A string that may contain text data related to the command. Typically not used.
lData	Long	A long integer value that may contain numeric data related to the command. Typically not used.

Parameter Requirements for Commands

Most commands do not require parameters. For example, `cmdbold` bolds (or unbolds) selected text, ignoring the `strTextData` and `lData` parameters.

For a list of standard commands and their parameters, see "Standard Commands" on page 199. You can also create custom commands to be executed programmatically. See "Custom Commands" on page 215.

The Toolbar Reset Command

Name: toolbarreset

Parameters:

Parameter	Type	Value if toolbar is freshly loaded from config.xml, not loaded from a saved configuration	Value if toolbar is generated by a reset
strTextData	string	NewLoad	FullReset
IData	long	1	0

Description: Resets the toolbar. For a complete explanation, see "Reacting to the Initialization of a Toolbar" on page 303.

Reacting to the Initialization of a Toolbar

When the Event is Sent to the Script

The ontoolbarreset event is sent under either of these conditions.

- When the editor first appears, a new toolbar is loaded. If no saved customization is found, the configuration data is read to build a fresh toolbar. Next, the ontoolbarreset event is sent to allow the script to add commands to the toolbar.
- When the user presses the Reset button on the Customize dialog, the toolbar is reset. When this happens, all old customizations are discarded, and the configuration data is read to create a new toolbar. At this time, the script can add commands by reacting to the ontoolbarreset event.

Script Reacting to a Toolbarreset Command

The ontoolbarreset event can be sent to a script using the addEventHandler method or the eWebEditProExecCommandHandlers array. If the scripting adds any buttons to the toolbar, the new toolbar configuration is saved in the user's customization settings, if allowed. (See "Letting Users Customize the Toolbar" on page 315.)

For example:

```
eWebEditPro.addEventHandler("ontoolbarreset", "loadStyleSheet(this.event.srcName)");
eWebEditPro.create("MyContent1", "100%", 400);
```

Using Toolbarreset to Reset Customization

The toolbarreset command can be sent by a script or defined in the configuration data when you want to quickly reset the toolbar features.

Script Implementing a Toolbarreset Command

You can use the toolbarreset command to reset a user's customization to a new XML configuration definition, even without changing the customization name in the configuration data.

To do this, save a cookie to the user's system. The script can check to see if the customization has happened. If it has not, the script could call the toolbarreset command to reset the named customization to the new XML definition. Then, the script could use the cookie to record that the update was done.

Implementing toolbarreset as a Toolbar Button

In the configuration data, you can assign a button to the toolbarreset command. If you do, the command executes when the user clicks the button. The command is passed on to the script after it executes, just as with standard commands.

No icon is assigned to the event, so you can choose any standard image. (See "Button Images" on page 360.)

WARNING! The toolbarreset command would be a dangerous toolbar button. The user could accidentally click it and reset everything. You may prefer to define your own external command, such as "jstoolbarreset", and implement it in the scripting instead of defining the command as a standard button. In this way, you can interact with the user appropriately and then send the ontoolbarreset event to the editor.

The Redisplay Toolbars Command

Name: redisplaytoolbars

Parameters: None

Description: Displays, or unhides, all toolbars. This command is only useful if the user removes all menus and cannot customize to get them back.

This command appears as the **Restore All Menus** item on the context menu *only* if the user cannot customize toolbars and the user has removed all menus from view.

See Also: "Letting Users Customize the Toolbar" on page 315

The Instance Object

The instance object is accessed via the instances collection of the **eWebEditPro+XML** object. For example:

```
var objInstance = eWebEditPro.instances[ "MyContent1" ]
```

This instance object has properties, methods, and events. You can also access the [instancetypes array](#).

For documentation of the methods, properties, and events, see "[Instances Object](#)" on page 9.

The onerror Event

Note that, for the instance object's onerror event,

- usually only the source property is available
- the event only fires if the save method fails

If the status of the **eWebEditPro+XML** object is EWEP_STATUS_SIZEEXCEEDED, two additional event properties are available to help troubleshoot the error.

- **contentSize** - the number of characters in the content
- **maxContentSize** - the maximum number of characters permitted, as specified by the maxContentSize parameter (See "[Property: maxContentSize](#)" on page 168.)

Here is an example of how to use these properties.

```
function myOnErrorHandler()
{
    if (EWEP_STATUS_SIZEEXCEEDED == this.status && "save" == this.event.source)
    {
        var strMsg = "HTML content size (in chars): " + this.event.contentSize +
                    " Maximum: " + this.event.maxContentSize;
        alert(strMsg);
    }
}

eWebEditPro.instances.MyContent1.onerror = myOnErrorHandler; //
```

Note that you cannot use `...onerror = "myOnErrorHandler()";`.

The instanceTypes Array

You can list all possible editor types by accessing the **eWebEditPro** JavaScript Object's `instanceTypes` array. For example:

```
document.write("Instance Types:<br>");
for (var i = 0; i < eWebEditPro.instanceTypes.length; i++)
{
    document.write(" &nbsp;" + eWebEditPro.instanceTypes[i].type + "<br>");
```

Also, you can use the `instanceTypes` Array to determine which type of editor to create. For example, to prevent the ActiveX-based editor from being created, assign a false value to this syntax:

```
instanceTypes["activex"].isSupported
```

If set to true, ActiveX is supported.

The Parameters Object

The parameters object is used to set parameters prior to creating an instance of the editor. This is a property of the **eWebEditPro+XML** object (for example, `eWebEditPro.parameters`).

See Also: “[Parameters Object](#)” on page 7

Use the parameters object to change default values for a particular instance of an editor. To change the default values for all instances of the editor, change the value in `ewebeditprodefaults.js`.

The names of most parameters match the names in `defaults.js`. The different ones are listed below.

NOTE

It is important to retain the case (upper or lower) of the letters when changing a parameter value.

default.js	parameters.
buttonTagStart	<code>.buttonTag.start</code>
buttonTagEnd	<code>.buttonTag.end</code>
popup*	<code>.popup.*</code> (the first letter after a popup becomes lowercase)

For a complete list of parameters and their `default.js` values, see “[Customizing the Popup Button](#)” on page 253.

Parameters Object Properties

These properties are the same as those in `ewebeditprodefaults.js`. In fact, `ewebeditprodefaults` initializes the parameters object.

The following parameters are part of the ActiveX control. Go to the listed page numbers to read about them.

- “[Property: hideAboutButton](#)” on page 162
- “[Property: Config](#)” on page 160
- “[Property: baseURL](#)” on page 150
- “[Property: CharSet](#)” on page 160
- “[Property: Title](#)” on page 163
- “[Property: bodyStyle](#)” on page 159

For additional Parameters Object properties, methods and events that are not part of the ActiveX control, see “[Parameters Object](#)” on page 7.

NOTE

Onblur, ondblclickelement, onexeccommand, and onfocus are events raised by the ActiveX control, not the parameters object. But you set them using the parameters object -- you cannot set them using the ActiveX control. As a result, they are documented as properties of the parameters object used to assign JavaScript that executes when the ActiveX control's event fires.

Installation Popup Window Defaults

See “[InstallPopup Object](#)” on page 10.

Popup Window Defaults

These defaults determine the attributes of the button that launches the popup window.

Examples

```
<script language="JavaScript">
var strhref = "JavaScript:";
if (eWebEditPro.isNetscape)
{
    strhref += "eWebEditPro.edit(\"MyContent1\")";
}

eWebEditPro.parameters.buttonTag.start = "<a href='" + strhref + "'><img alt=Edit width=150
height=60 src=button.gif>

eWebEditPro.parameters.buttonTag.value = "";
eWebEditPro.parameters.buttonTag.end = "></a>";

</script>
```

See Also: “[Popup Object](#)” on page 11

eWebEditProUtil JavaScript Object

The eWebEditProUtil JavaScript object offers utility functions. A file, eweputil.js, offers helpful functions and properties by way of the eWebEditProUtil JavaScript object.

You can use the eWebEditProUtil object in a Web page that includes ewebeditpro.js or eweputil.js.

- Pages that display the editor must include ewebeditpro.js
- Popup pages that do not display the editor but access it must include eweputil.js to use the eWebEditProUtil object

To review a sample that uses eWebEditProUtil, see formelementinsert.htm, located in the **eWebEditPro+XML** folder.

For a page that does *not* display the editor, add the following include to create eWebEditProUtil.

```
<script language="JavaScript1.2" type="text/javascript" src="eweputil.js"></script>
```

The eWebEditProUtil JavaScript object has several properties and methods. To learn about them, see "[eWebEditProUtil Object](#)" on page 4.

ActiveX Control

Web masters can exert control over **eWebEditPro+XML**'s functionality and content through modifying the ActiveX control properties and methods.

This section explains the properties, methods and events of the **eWebEditPro+XML** ActiveX control. It covers the following topics.

- [Accessing the ActiveX Control](#)
- [ActiveX Properties, Methods and Events](#)

Accessing the ActiveX Control Using JavaScript

There are several ways to access the **eWebEditPro+XML** ActiveX control using JavaScript. Choose the methods that are most convenient for your situation.

Do not confuse the ActiveX control with the **eWebEditPro+XML** JavaScript object and the Instance JavaScript object. The JavaScript objects wrap the ActiveX control, making it very easy to integrate into a Web page. Without them, a developer would need to write the integration code that moves content in and out of the editor, detects the browser, and displays a textarea field if the ActiveX control is not supported.

See "[ActiveX Properties, Methods and Events](#)" on page 310 to learn about methods, properties, and events associated with the ActiveX control.

See "[JavaScript Objects](#)" on page 300 to learn which methods, properties and events are associated with JavaScript objects.

eWebEditPro+XML JavaScript object

The **eWebEditPro+XML** JavaScript object is accessed directly in JavaScript. It is a single object that is automatically created when a Web page includes the ewebeditpro.js file.

IMPORTANT!

The **eWebEditPro+XML** JavaScript object is *not* the ActiveX control. It is required to access the ActiveX control, but the ActiveX methods (for example, pasteHTML) are not methods of the **eWebEditPro+XML** JavaScript object.

```
<script language="JavaScript1.2">
  eWebEditPro
</script>
```

See Also: "[eWebEditPro Object](#)" on page 4

eWebEditPro+XML ActiveX control

Below are examples how to access the **eWebEditPro+XML** ActiveX control in JavaScript. In these examples

- the name of the editor is "MyContent1"

- the JavaScript variable 'sEditorName' is presumed to hold the name of the editor, as in `sEditorName = "MyContent1"`
- the JavaScript variable 'i' is presumed to be a valid numeric index

```
<script language="JavaScript1.2">
  eWebEditPro.instances.MyContent1.editor
  eWebEditPro["MyContent1"]
  eWebEditPro.instances[sEditorName].editor
  eWebEditPro.instances.MyContent1.editor
  eWebEditPro.instances["MyContent1"].editor
  eWebEditPro.instances[sEditorName].editor
  eWebEditPro.instances[0].editor
  eWebEditPro.instances[i].editor
</script>
```

See Also:

- "Property: instances collection" on page 178 for more on the instances array
- "eWebEditPro ActiveX Control Object" on page 13

Instance JavaScript object

The Instance JavaScript object is actually an array of objects. Each instance of the editor on a page is represented by an instance object. The array may be indexed by a number or string name of the editor, or (as with all JavaScript arrays) may be identified by name, separated by a period (.).

See Also: "Instances Object" on page 9

NOTE

The ActiveX control is accessible from the Instance object by using the editor property. For example, `objInstance.editor`.

In these examples, the name of the editor is "MyContent1". The JavaScript variable 'sEditorName' is presumed to hold the name of the editor, as in `sEditorName = "MyContent1"`. Likewise, the JavaScript variable 'i' is presumed to be a valid numeric index.

```
<script language="JavaScript1.2">
  eWebEditPro.instances.MyContent1
  eWebEditPro.instances["MyContent1"]
  eWebEditPro.instances[sEditorName]
  eWebEditPro.instances[0]
  eWebEditPro.instances[i]
</script>
```

See Also: Ektron Knowledge Base article "JavaScript Error Accessing Editor Name" (http://www.ektron.com//support/ewebeditprokb.cfm?doc_id=1200)

ActiveX Properties, Methods and Events

You can modify the values for the default ActiveX control properties in the `ewebeditprodefaults.js` file, using a standard text editor such as Notepad.

You can also modify ActiveX control property values for individual instances of the editor. See “[The Parameters Object](#)” on page 306.

For details on the properties, methods and events, see “[eWebEditPro ActiveX Control Object](#)” on page 13.

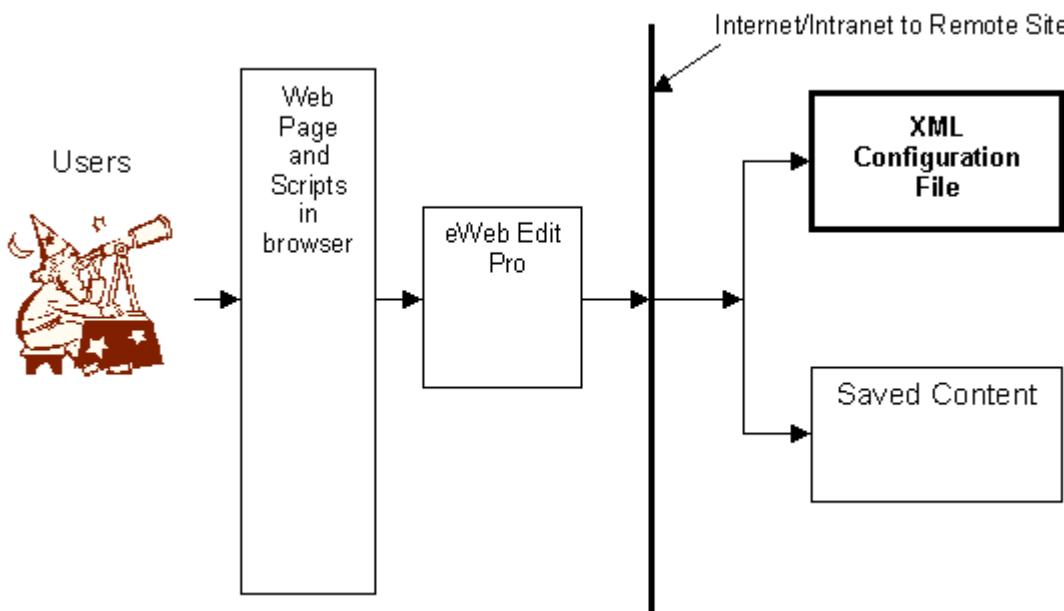
The Configuration Data

eWebEditPro+XML's configuration data lets you define many aspects of editor functionality. For example, by modifying the configuration data, you can

- enable/disable features, such as automatic spell check
- arrange toolbars
- add custom commands
- determine whether users can edit HTML source code
- manage the image selection feature

Managing the Configuration Data

The Site Administrator controls the configuration data and specifies which configuration data to use. Users cannot edit the configuration data.



Editing the Configuration Data

To implement a standard configuration of **eWebEditPro+XML**, leave the configuration data as is. If you want to modify the configuration data, you have two choices:

- change the configuration data dynamically (see “[Dynamically Changing the Editor](#)” on page 250)

- edit the config.xml file using your favorite text editor or a specialized XML editor (continue reading this section)

If you edit the config.xml file, be very careful to adhere to the format. For example, if you accidentally delete a less than character (<), your edits are not applied.

XML is case-sensitive. Therefore, keep all element names (for example, <command>) and attribute names (for example, name) lower case.

If you use an XML editor to edit config.xml, Ektron supplies a corresponding schema file (config.xsd) that can validate config.xml. By default, the config.xsd is installed to the ewebeditpro5 directory. Note that some validators might find errors when validating config.xml against config.xsd because some attributes have no value by default.

If you want to insert HTML as a stream into the config.xml file (as opposed to as a file specification), delete the encoding attribute information (`encoding=`) at the top of the file.

NOTE

If you are using **eWebEditPro+XML** within an Ektron Content Management System (CMS), and you want to modify the configuration data via a file, you can find samples of the file within the folder.

Customizing Configuration Data for Data Designer Content

The config.xml settings (described above) check regular HTML content. If you are using the Data Designer, you must make the same changes to the configdataentry.xml file.

Providing Configuration Files for User Groups

Since the file is designated at run time, you can use scripting to determine which configuration data the user loads. The following are a few options for implementing separate configuration files for different user groups in your organization.

In this example, you create a configuration file named admin.xml.

- Set up a series of Web pages for each group to log into. Each page specifies which configuration data to use.

For example, you could change the configuration data in the HTML page that launches the editor using this line.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.parameters.config="administration.xml"
eWebEditPro.create("MyContent2", 700, 250);
//-->
</script>
```

(See Also: “[Appendix A: Naming the eWebEditPro+XML Editor](#)” on [page 778](#).)

You can set up and reference different file names or different file locations. Using different file names is probably easier if you are starting with the sample files provided by Ektron.

- Use the user's login name to determine which configuration data to use. In an ASP or ColdFusion environment, you can use the login name as a search key in a database to retrieve the configuration data that the user should access.
- Use the login name as the XML file name. You can keep all configuration files in one location and build the xml file name using the login name.

```
strCfgFile = "http://www.ektron.com/configs/" +
Login.value + ".xml
ewebeditpro1.Config = strCfgFile
```

This is similar to a user's profile that is set up when someone logs into an operating system.

If the user does not have a profile, the user gets the editor's default functionality.

Changing the Configuration Data's Location

The location of the configuration data is specified in the ewebeditprodefaults.js file, which is located in the folder to which you installed **eWebEditPro+XML**.

The configuration data's location is specified in the `this.config =` attribute. To change the location of the configuration data on the client, edit this line.

Troubleshooting Problems with the Configuration Data

Sometimes, when you change the configuration data, you refresh the page that hosts **eWebEditPro+XML** but still cannot see the effect of those changes. For example, you add buttons to the `<interface>` section of config.xml, but the toolbar does not display the buttons when **eWebEditPro+XML** loads.

For possible solutions to this problem, see "[Changes to config.xml Have No Effect](#)" on page 317.

Organization of Configuration Documentation

Documentation for the configuration data consists of the following topics.

- ["Defining the Toolbar"](#) on page 230
- ["Letting Users Customize the Toolbar"](#) on page 315
- ["Overview of Configuration Data"](#) on page 319

Letting Users Customize the Toolbar

The `allowCustomize` attribute of the `interface` element is part of the configuration data. Possible values are **true** and **false**.

```
<interface name="beta" allowCustomize="true">
```

Modify this attribute to let users customize **eWebEditPro+XML**. If the attribute is set to “**true**”, users can

- create a new toolbar menu
- remove an existing toolbar menu
- add commands to a toolbar menu
- remove commands from a toolbar menu
- rearrange the commands on a toolbar menu

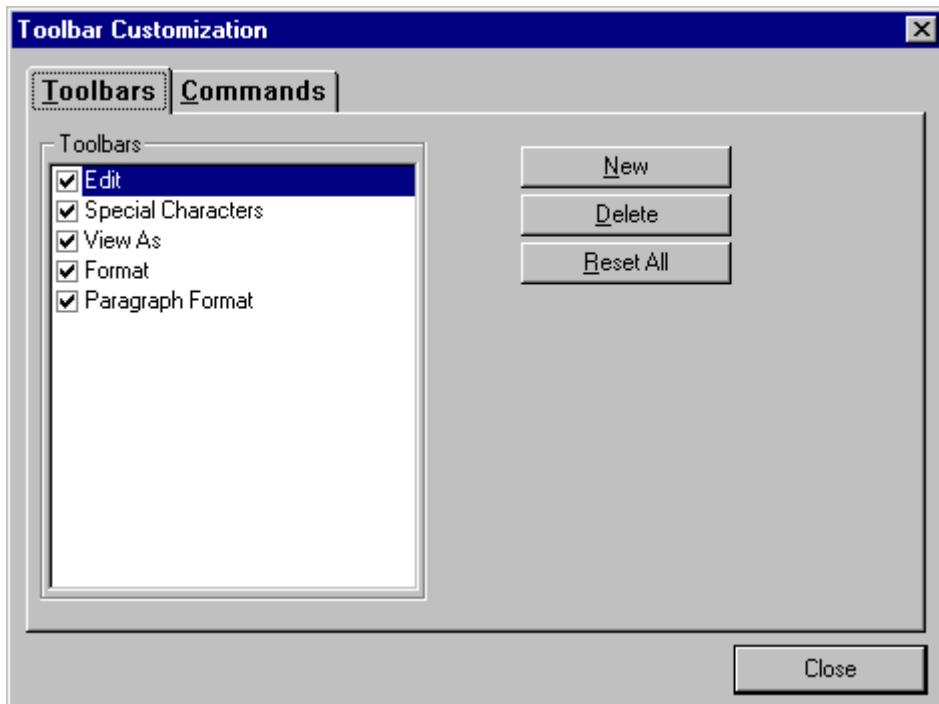
NOTE

Users can only add commands defined in the configuration data to a toolbar menu.

If `allowCustomize` is set to **false**, the **Customize** option does not appear on the user’s customization menu (the menu that the user invokes the customize the toolbar).

Allowing User Customization

The user places the cursor on a toolbar and right clicks the mouse to invoke the customize dialog box (illustrated below).



After the user customizes toolbar menus and presses **Close**, the customization files are saved in the client PC's temporary folder. The file's names (prior to the file extensions) match the value assigned to the `interface name` attribute in the configuration data.

The next time the user opens that page, the customized toolbar appears. From this point on, any changes you make to the interface section of the configuration data on the server are not used on the user's computer.

If you want to apply changes to the interface section of the configuration data to all users, see "[Overriding User Customization](#)" on page 317.

Preventing Customization by Users

If you set `allowCustomize` to **false**, users cannot permanently customize their toolbars. The system uses the default toolbar and menu specifications defined in the configuration data.

NOTE

If you set `allowCustomize` to **false**, the user still sees the customize option, and customization procedure acts the same. However, the customization is only saved while the user remains on the page. Once the user leaves the page, the customization is lost.

Overriding User Customization

You might allow users to customize **eWebEditPro+XML**, but later need to implement a global change to the editor. For example, you may decide that users cannot edit HTML code.

To override all user customization, follow these steps.

1. Make the necessary changes in the configuration data on the server.
2. Change the value of the `interface name` attribute. For example, if the attribute's value is **beta**, you could change it to **beta1**.

WARNING!

If you override user customization, users lose all changes made to **eWebEditPro+XML** toolbars and menu configurations. If the users preferred those customizations, they must redo them.

To understand how changing the `interface name` attribute of the configuration data updates all user configurations, read “[Determining Which Configuration Data to Use](#)” on page 317.

Determining Which Configuration Data to Use

When a user launches **eWebEditPro+XML**, the following events occur.

1. The browser reads the configuration data on the server to determine which data to use. The file name is the value of the `interface name` attribute.
 2. The browser looks for a customization file with that name in the temporary folder on the user's computer. If it finds one, that configuration data determines which toolbars to display.
- If the browser does not find a customization file on the user's computer, it defaults to the interface section of the configuration data on the server to determine which toolbars to display.

Changes to config.xml Have No Effect

Sometimes, you might change the configuration data but the changes have no effect. For example, you add buttons to the `<interface>` section of the configuration data, but the toolbar does not display them when **eWebEditPro+XML** loads in the browser.

This table suggests how to fix this problem.

Possible Cause	Resolution
Wrong configuration file - You did not modify the configuration file that is being loaded. It is quite common to edit the wrong file.	Double check all the paths and the config parameter to ensure that you are using the correct configuration. Search for all config.xml files on the server.

Possible Cause	Resolution
Cache - Either the browser cached (that is, stored on your PC) an old version of the page displaying the editor, or the Web server is returning an older version from its cache.	<p>Enter the URL of the configuration file into your browser's address bar (this works best with Internet Explorer 5.0 or later). The configuration data should appear. Ensure that your changes are present.</p> <p>If they are not, clear the browser cache. In IE, from the Tools -> Options dialog, delete temporary Internet files.</p> <p>If the problem persists, force the Web server to read the file by placing ?x=1 at the end of the URL, as shown here:</p> <p>Browse to “..../config.xml?x=1”.</p> <p>If the correct file appears, the Web server has cached the file. Either restart the server or wait for the server to refresh the cache.</p>
Customized by user - If the end user customized the toolbar (which can only occur if the allowCustomize attribute of the <interface> element is “true”), changes to the configuration data are not applied.	<p>The editor will display the toolbar specified in the configuration data after one of these events occurs:</p> <ul style="list-style-type: none"> • The user opens the customize dialog and clicks the Reset All button. This causes the editor to display the toolbar according to the configuration data. • The interface element name is changed in the configuration data. For example: <pre><interface name="custom2" ...></pre> <p>TIP: To ensure that interface is updated, name it with the date and time the configuration was changed. For example, <interface name="custom_20010824_1318" ...>.</p> <ul style="list-style-type: none"> • The user is prevented from customizing the toolbar. To do this, set the allowCustomize attribute to “false” as shown: <pre><interface ... allowCustomize="false"></pre>

Overview of Configuration Data

This section presents two charts that depict the configuration data:

- a functional view, which arranges configuration data by task
- a hierarchical view, which arranges configuration data by XML element

If you are reading this online, you can click on purple items to get more information.

Following the illustration is a table that describes the major components in alphabetical order. The table links to more detail about each configuration element.

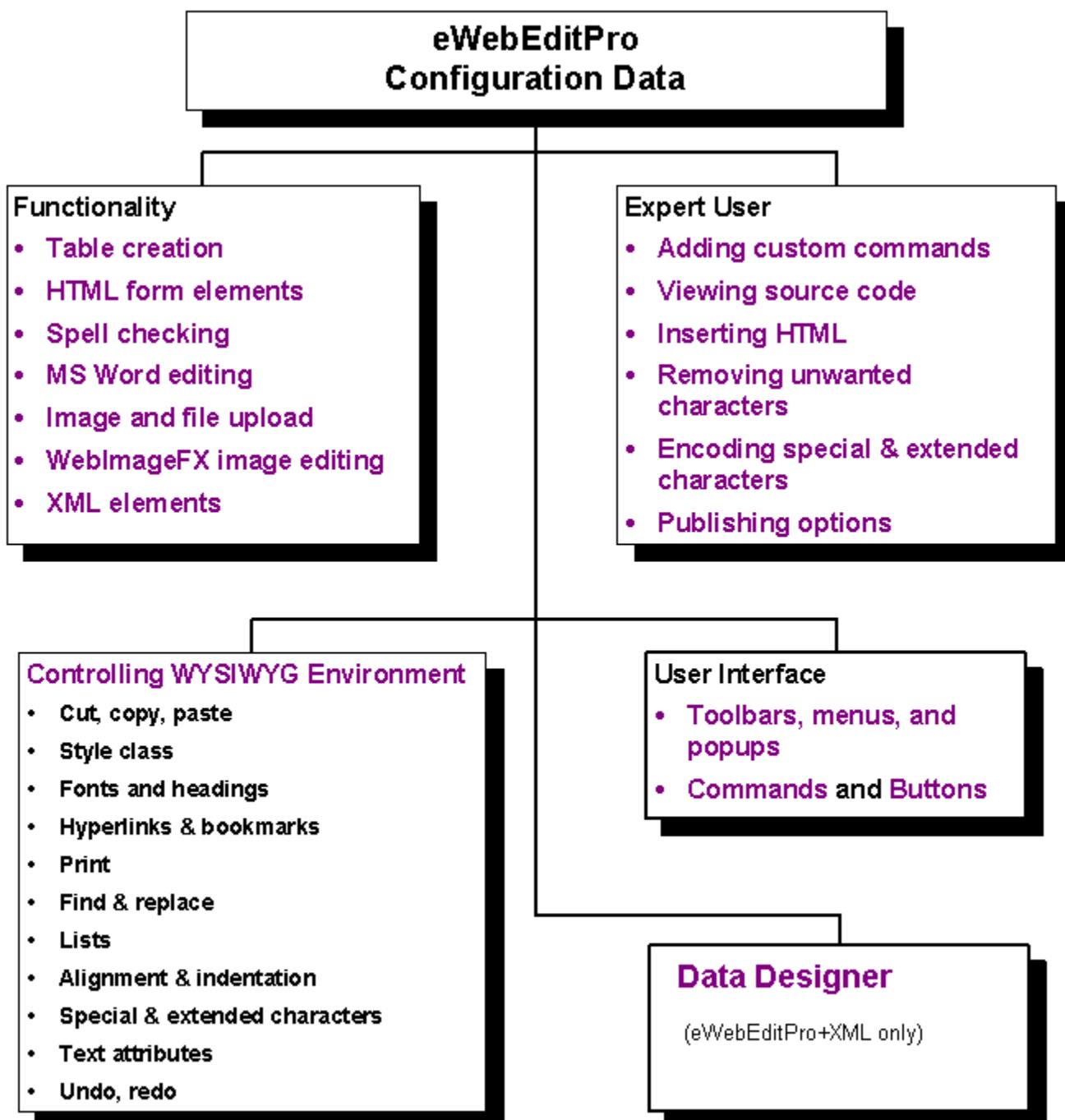
NOTE

You can also edit configuration data dynamically. For information, see “Dynamically Changing the Editor” on page 250.

NOTE

If you use an XML editor to edit config.xml, Ektron supplies a corresponding schema file (config.xsd) that you can use to validate config.xml. By default, the config.xsd is installed to the `ewebedit\pro5` directory. Note that some validators might find errors when validating config.xml against config.xsd because some attributes have no value by default.

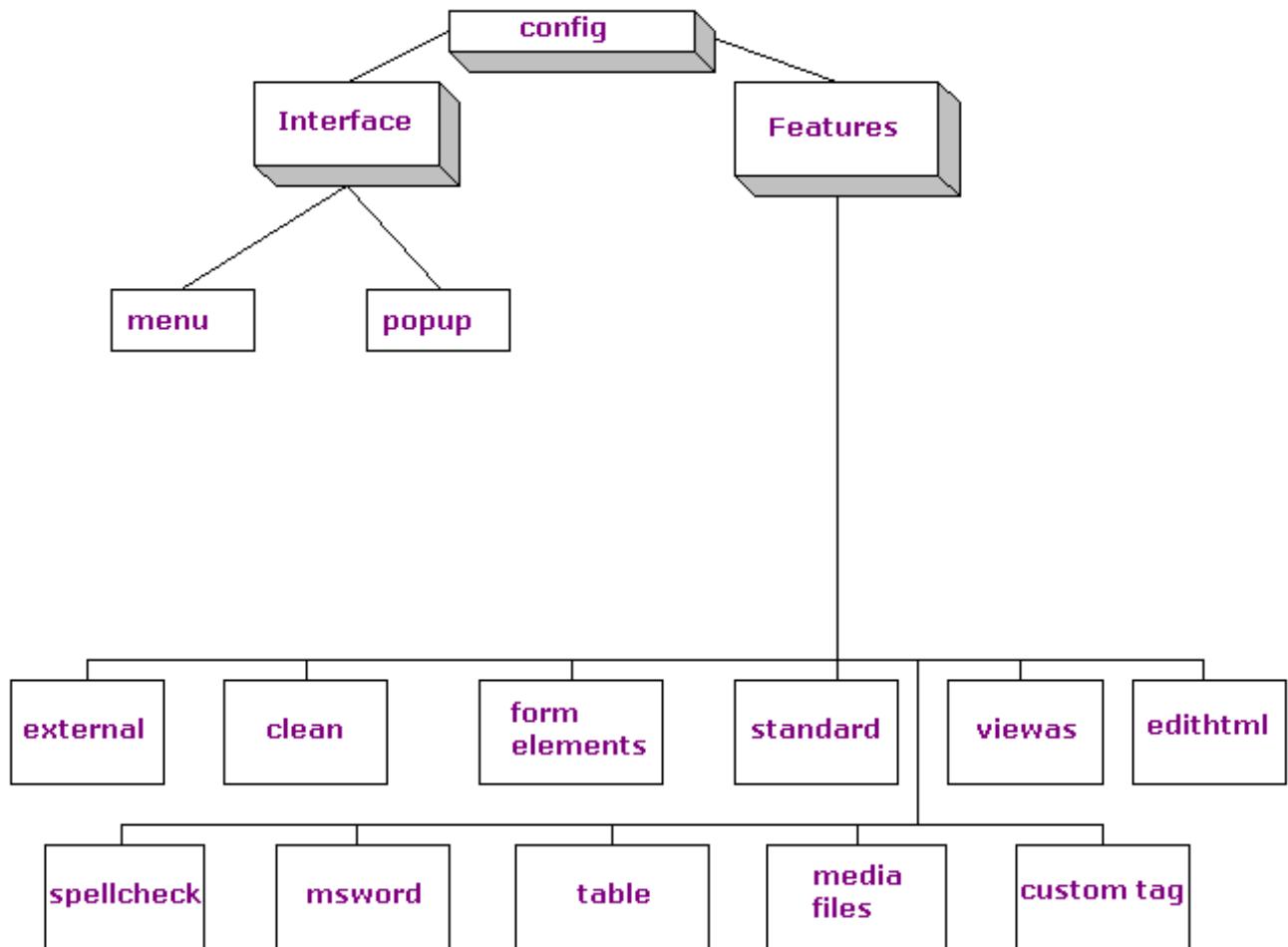
Configuration Data: Functional View



Configuration Data: Functional View Topic List

Topic	For more information, see
Functionality	
Table creation	"Managing Tables" on page 371
HTML form elements	"Form Elements" on page 388
Spell checking	"The Spellcheck Feature" on page 404
MS Word editing	"Editing in Microsoft Word" on page 411
Image and file upload	"The Mediafiles Feature" on page 493
WebImageFX image editing	"Imageedit element" on page 508
XML elements	"The XML DTD and Schema" on page 678
Expert User	
Adding custom commands	"External Features" on page 386
Viewing source code	"The ViewAs Feature" on page 390
Inserting HTML	"The EditHTML Feature" on page 392
Removing unwanted characters	"Cleaning HTML" on page 393
Encoding special & extended characters	"Configuring for Extended and Special Characters" on page 419
Publishing options	"publish" on page 355
Controlling WYSIWYG Environment	"Standard Commands" on page 199
Toolbars, menus, and popups	"User Interface Element Definitions" on page 332
Commands	"Commands" on page 195
Buttons	"Button Images" on page 360

Configuration Data: Hierarchical View



Configuration Elements in Alphabetical Order

Element	Function	For more info, see	Example
autoupload	Defines the automatic image upload mechanism.	"Autoupload Element" on page 498	
bar	Places vertical or horizontal bar on menu	"bar" on page 332	<bar/>

Element	Function	For more info, see	Example
button	Defines toolbar button or menu option	"button" on page 333	<button command="cmdcut" />
caption	Describes a menu bar or toolbar button in the user interface	"Caption" on page 335	<caption localeRef="btnTxtVAHtml">View As HTML </caption>
clean	Ensures that content is readable and concise HTML	"Cleaning HTML" on page 393	<clean cr="cr" lf="lf"/>
cmd	An abbreviated version of <command>.	"cmd" on page 339	<cmd name="cmdprint" key="print" ref="sPrint" />
command	Defines a standard editor action, such as copying text.	"command" on page 336	<command name="cmdviewashtml" style="icon" visible="true"></command>
config	Identifies the root element of the file	"The Config Element" on page 326	<config product="eWebEditPro">
customtag (only appears if you are using eWebEditPro+XML)	Lets you define the appearance of XML elements and attributes	"The Custom XML Tag DTD and Schema" on page 678	<tagdefault type="vertical" visible="true" style="font-family:arial;">
docxml	Lets you configure eWebEditPro+XML to insert an element's required elements and attributes	"Docxml Element" on page 696	
domain	The domain name for the connection.	"Domain Element" on page 503	
editHTML	Determines whether users can edit HTML source code	"The EditHTML Feature" on page 392	<edithtml enabled="false"/>
external	Defines external client functionality	"External Features" on page 386	<external enabled="true">
features	Defines standard and custom commands	"The Features Element" on page 327	<features>
formelements	Let the user create an HTML form	"Form Elements" on page 388	<form name="Test" action="http://localhost/ewebeditpro5/formtest.htm" method="post"></form>
glyph	A glyph, or icon, that can represent custom tag to the user	"Glyph Element" on page 693	

Element	Function	For more info, see	Example
imageedit	Defines location of Ektron WebImageFX image editor.	"Imageedit element" on page 508	<pre><imageedit> <control src="[WebImageFXPath]/ ImageEditConfig.xml" /> </imageedit></pre>
interface	Lets you define the user interface	"Letting Users Customize the Toolbar" on page 315, "The Features Element" on page 327	<pre><interface name="beta4" allowCustomize="false"></pre>
listchoice	Defines an individual choice in a list box command item	"listchoice" on page 345	<pre><selections name="headinglist" enabled="true" sorted="true"> <listchoice>Arial, Helvetica</listchoice> </command></pre>
loadsch	A list of schemas to load	"Loadsch Element" on page 699	
math	Controls the math expression editor	*****	<pre><math imagetype="png"> <cmd name="cmdmath" key="math" ref="cmdMath"/> <toolbar></toolbar> </math></pre>
maxsizek	Specifies maximum file size of upload.	"Maxsizek Element" on page 496	
mediaconfig	Controls the operation of the configuration dialogs	"Mediaconfig Element" on page 496	<pre><mediaconfig allowedit="true" /></pre>
mediafiles	Controls the selection and upload of media (for example, images)	"The Mediafiles Feature" on page 493	<pre><mediafiles> <command name="cmdmfumedia" style="icon" visible="true"> </mediafiles></pre>
menu	Defines a toolbar or pulldown menu	"menu" on page 349	<pre><menu name="editbar" newRow="false" showButtonsCaptions="false" wrap="false"> <caption LocaleRef="btnMainCap">Edit </caption></pre>
msword	Lets you edit within Microsoft Word	"Editing in Microsoft Word" on page 411	<pre><msword enabled="true"> <cmd name="cmdmsword" key="msword" ref="cmdMSW" style="toggle"/> </msword></pre>

Element	Function	For more info, see	Example
password	Provides the password for gaining access to the server.	"Password Element" on page 502	
popup	Defines a popup menu	"popup" on page 351	
port	Specifies which port to use for any file transfers.	"Port Element" on page 506	
resolvemethod	Defines how to resolve file paths	"Resolvemethod Element" on page 507	
selections	Defines a list of items within a listchoice command.	"listchoice" on page 345	see listchoice
simtaglist and simtag	Reduce maintenance of XML data for tags with similar attributes	"Simtaglist Element" on page 694; "Simtag Element" on page 695	
spellayt	Defines how spell checking as-you-type operates	"Spellayt" on page 406	<spellayt autostart="false" markmisspelledsrc="[eWebEditProPath]/wavyred.gif" delay="20" />
spellcheck	Controls the operation of spell checking	"The Spellcheck Feature" on page 404	<spellcheck enabled="true">
spellingsuggestion	Suggestions for correcting errors when using spell checking "as-you-type	"Spellingsuggestion" on page 407	<spellingsuggestion enabled="false" max="4" />
standard	Defines standard editing commands and options	"standard" on page 354	<standard autoclean="true" publish="xhtml">
style	Defines style sheet implementation	"style" on page 357	
table	Allows users to create tables	"Managing Tables" on page 371	<table enabled="true">
tagatrdlg	Controls the Custom Tag attribute dialog	"tagatrdlg" on page 702	
tagdefault	Default attribute values that determine a tag's appearance if a tag is not defined.	"Tagdefault Element" on page 693	
tagdefinitions	Affects overall functionality of custom tags feature.	"Tagdefinitions Element" on page 683	
taginsdlg	Controls the Insert Custom Tag dialog	"taginsdlg" on page 701	
tagpropdlg	Controls the Tag Properties dialog	"tagpropdlg" on page 703	

Element	Function	For more info, see	Example
tagspec	Specify appearance of custom tag	"Tagspec Element" on page 684	
tooltiptext	Defines text that appears when cursor hovers over an icon	"toolTipText" on page 358	<toolTipText localeRef="bttn1">Align Left</toolTipText>
transform	Specifies transformation files to use when loading or saving content.	"Transform Element" on page 698	
transport	Defines the mechanism used to select and upload media files.	"Transport Element" on page 497	<transport allowupload="true" type="post" xfer="binary" pasv="true">
username	Provides the user name for gaining access to the server.	"Username Element" on page 502	
validext	Valid file extensions allowed for upload	"Validext Element" on page 495	<validext>gif,jpg,png,jpeg,jp e</validext>
viewAs	Determines whether users can view HTML source code	"The ViewAs Feature" on page 390	<viewas enabled="true" publish="xhtml" mode="whole">
webroot	Specifies the path to use when referencing an uploaded file.	"Webroot Element" on page 505	
xferdir	The destination directory on the server for the upload	"Xferdir Element" on page 504	
xml Declaration	Identifies the file as an xml file		<?xml version="1.0" encoding="iso-8859-1"?>
xsd	Maintains a list of schemas to load	"XSD Element" on page 700	

The Config Element

Config is the root element that contains all information about the elements of **eWebEditPro+XML** that you are defining. All other elements are defined within the config element.

Therefore, you create different configuration data for every unique set of functions that you are implementing. For example, if one user group can view source code, while another group cannot, you would create two sets of configuration data.

Users can customize **eWebEditPro+XML** toolbars. See "Letting Users Customize the Toolbar" on page 315 for details.

The Interface Element

Use the interface section of the configuration data to define the user interface. Within the interface element, you can modify

- which toolbar buttons are available to the user

- the sequence of toolbar buttons
- space between toolbar options

NOTE A toolbar button typically executes a command. Commands are defined in the Features element.

Buttons not Assigned to Menus

By default, some commands are not assigned to any standard menu. However, the user can place any enabled command on a menu. This procedure is explained in the **eWebEditPro+XML** User Guide's section "Removing or Adding Menu Items."

The Features Element

Use the features section of the configuration data to define the commands that are assigned to buttons and menus in the interface section. You can

- delete standard commands
- add custom commands
- modify the images and text that appear with the command on a toolbar button or menu
- if the command's style is list, enter the listchoice items on the list
- set feature options, such as enabling publishing options

Attribute Types

Each element has one or more attributes that let you tailor its function to your unique needs. Each attribute is one of the three types listed below.

The attributes are actually always strings, but the editor expects their values to be one of the types listed below.

Boolean

These are the valid string values for Boolean attributes.

Positive	Negative
yes	no
true	false
1	0
ok	[unknown]

Integer

An attribute that is expected to contain numeric values is interpreted as an integer. If an integer value contains alpha characters, it is converted to 0.

String

An attribute interpreted as this type uses the text given without interpretation. All characters are converted to lower case unless the text is defined as a path.

User Interface Elements: Standard, Menu, and Popup

The configuration data contains several elements that let you define the **eWebEditPro+XML** toolbar. For example, the button element lets you define the image that appears on a toolbar button, and the command that is executed when the user presses the button.

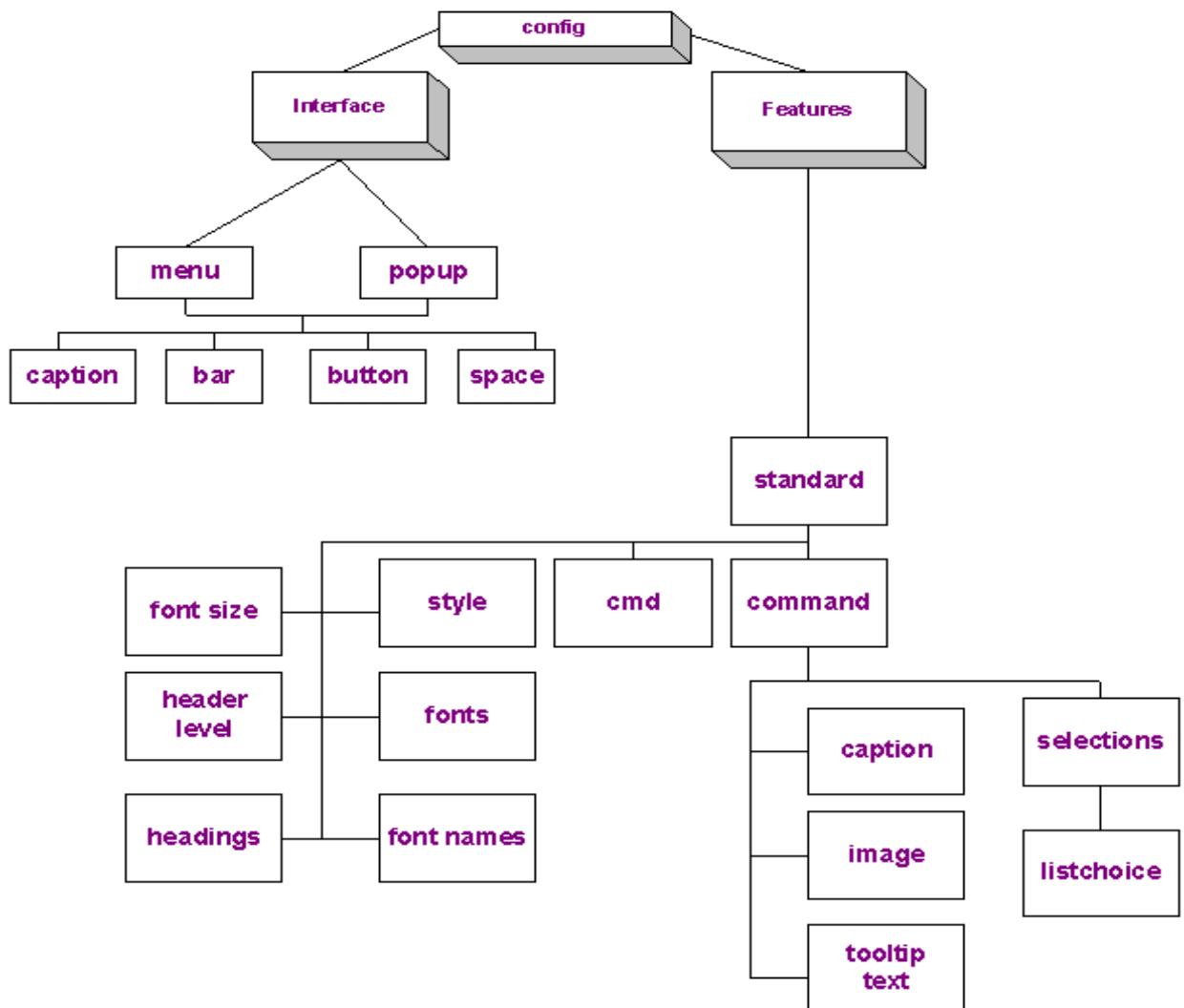
NOTE

You can also edit configuration data dynamically. For information, see “Dynamically Changing the Editor” on page 250.

The following chart illustrates the main config.xml elements that let you determine **eWebEditPro+XML**’s user interface. Following the chart is a table that lists the components in alphabetical order.

Review the chart and table for an overview of these components, then proceed to subsequent sections for details about each component.

User Interface Element Hierarchy



User Interface Elements in Alphabetical Order

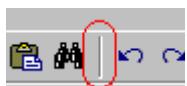
Element	Description	For more information, see
bar	Separates a group of commands from other commands on a menu.	"bar" on page 332
button	A toolbar button or menu item.	"button" on page 333
caption	Text describing a menu bar or toolbar button.	"Caption" on page 335
command	A standard or custom editor action, such as copying text.	"command" on page 336
cmd	An abbreviated version of <command>.	"cmd" on page 339
config	The single root element that signifies that this configuration belongs to eWebEditPro+XML .	"config" on page 340
features	One of the two major sections of the configuration data. Defines all standard and custom commands, and publishing options.	"features" on page 341
font	Specifies font names and sizes	"fonts" on page 380
font size	Specifies font sizes	"fontsize" on page 381
font name	Specifies font names	"fontname" on page 380
header level	Specifies available heading levels for paragraphs	"heading[X]" on page 383
headings	Defines a heading level	"headings" on page 382
image	An image to display on a button.	"image" on page 342

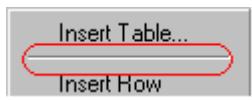
Element	Description	For more information, see
interface	One of two major sections of the configuration data. Defines toolbars, menus, dialogs, and other interface items.	"interface" on page 343
listchoice	Each item on a list.	"listchoice" on page 345
menu	A toolbar or pulldown menu.	"menu" on page 349
popup	A menu that is launched by pressing a toolbar button.	"popup" on page 351
selections	A group of listchoice items.	"selections" on page 352
space	A separator between toolbar buttons or popup menus.	"space" on page 353
standard	Standard editing commands and options.	"standard" on page 354
style	Defines style sheet and other aspects of style sheet implementation.	"style" on page 357
toolTipText	Text that appears when the cursor hovers over a toolbar button.	"toolTipText" on page 358

User Interface Element Definitions

bar

Places a

- vertical bar  on a toolbar or

- horizontal bar  on a [popup menu](#)

The bar separates one or more [commands](#) from other commands on a menu.

See Also: “Adding a Separator Bar Between Two Toolbar Menu Items” on page 241

NOTE Unlike the other commands, the bar and space elements are not defined in the configuration data. You cannot modify their appearance.

Element Hierarchy

```
<config>
  <interface>
    <menu>
      <bar>
```

```
<config>
  <interface>
    <popup>
      <bar>
```

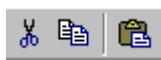
Attributes

Name	Attribute Type	Default	Description
None			There are no attributes to the bar element.

Example

```
<menu name="editbar">
  <caption visible="false" localeRef="btnMainCap">Edit</caption>
  <button command="cmdcut" />
  <button command="cmdcopy" />
  <bar/>
  <button command="cmdpaste" />
</menu>
```

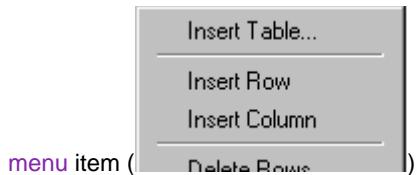
The result of this code would look like this.



button



Defines either a toolbar button () or a



menu item ().

The button element has a `command` attribute that identifies a command to execute when the user selects a toolbar button or a menu item. The value assigned to the button's `command` attribute must be defined within a command element. If not, the command is not added to the icon bar or menu item.

The order in which button elements are entered within a menu or popup menu command determines the order in which menu items appear on icon bars or menus.

See Also: “Adding a Toolbar Button” on page 237 and “Removing a Toolbar Button or Dropdown List” on page 240

Element Hierarchy

```
<config>
  <interface>
    <menu>
      <button>

<config>
  <interface>
    <popup>
      <button>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the command enabled? If false, the button is grayed.
popup	String	“”	Defines a Popup menu to display when the button is selected. (See “ popup ” on page 351.) If a popup is defined, the command name is not sent to the client.
command	A command element	“”	The command to execute when the user clicks the button. <i>See Also:</i> “ Commands ” on page 195

Example

```
<menu name="samplebar">
```

```

<caption visible="false" localeRef="btnMainCap">Sample</caption>
<button command="cmdcut" />
<button command="myselections" popup="myPopup" />
</menu>

```

Caption

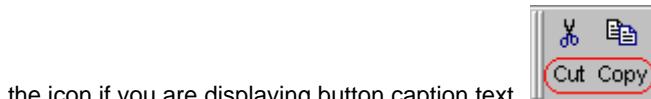
Provides the text that describes a [menu bar](#) or [toolbar button](#) in the user interface.

If a caption is assigned to a menu, the caption text only appears when the menu



bar is floating.

If a caption is assigned to a button, the caption text appears on the toolbar with



the icon if you are displaying button caption text.

NOTE The `textAlignment` attribute of the `menu` element determines the alignment of text within a button.

See Also: “[Creating or Editing the Toolbar Menu Caption](#)” on page 236 and “[Displaying Button Caption Text](#)” on page 243

Element Hierarchy

```

<config>
  <interface>
    <menu>
      <caption>

<config>
  <features>
    <standard>
      <command>
        <caption>

```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the element enabled?
localref	String	""	Localization identifier. This value translates the caption into a local language. See Also: "Translating Button Captions and Tool Tips" on page 244
visible	Boolean	Yes	Is the caption visible by default?

Examples

Button

```
<command name="cmdviewashtml" style="icon" visible="true">
<caption localeRef="btnTxtVAHtml">View As HTML </caption>
```

Menu

```
<menu name="editbar" newRow="false" showButtonsCaptions="true"
      textAlign="bottom" wrap="false">
    <caption visible="true" localeRef="btnMainCap">Edit</caption>
</menu>
```

command

Defines a standard editor action, such as copying text. Before commands can be used by a toolbar **button** or **menu** item, they must be defined and enabled.

A command definition does not need to be directly under a **feature** element. But it must be contained somewhere within a feature hierarchy.

See Also: "Commands" on page 195 and "cmd" on page 339

Element Hierarchy

```
<config>
  <features>
    <standard>
      <command>
```

Attributes

Name	Attribute Type	Default	Description
caption	String	""	Provides the text that describes a menu bar or toolbar button in the user interface. See Also: "Caption" on page 335
enabled	Boolean	Yes	Is the command enabled? If false, the command is not created.
image	String (key and/or src)	""	The image that appears if the command is assigned to a button. See Also: "Button Images" on page 360
maxwidth	Number	10	The maximum number of characters wide to make a command. This attribute only applies when the command is a list box or edit box.
name	String	""	The command's name. The name must be unique.
ref	String	""	Can replace caption or toolTipText, or both. Enter a code from the localization file to define a command's caption or ToolTiptext or both. See Also: "Translating Button Captions and Tool Tips" on page 244
selections	String	""	Defines a list of items within a listchoice command. See Also: "selections" on page 352
style	String	default	The style of the command when it appears on an icon bar or menu. The command can be one of these styles. <ul style="list-style-type: none"> • "icon" • "toggle" • "listbox" • "edit" For more information, see "Command Styles" on page 338.

Name	Attribute Type	Default	Description
toolTipText	String	""	Defines the tool tip text that pops up when the cursor hovers over an icon. See Also: "toolTipText" on page 358
visible	Boolean	True	Is the command visible when first created? If set to false, the command is created but is not displayed by default.

Command Styles

The command can be one of these styles.

Style	Values that indicate this style in style attribute	Description
"icon"	default, icon, or unknown	A toolbar button that is drawn as a rectangle normally containing an image. The button can contain both an icon and a caption, just the icon, or just the caption.
"toggle"	toggle	A button that maintains a pressed or checked state. If shown in a list box, it displays with a check. If shown on a toolbar, it is drawn as an 'icon' style command but is pressed in when checked. Clicking on the item toggles its state between check and unchecked or pressed in and popped out. If drawn on a toolbar, it is drawn using the same options as the 'icon' style.
"listbox"	listbox, list	This creates a command button that is displayed as a dropdown list box. The items for this listbox are defined in the selections element which is contained within the command element defining the command. (See "listchoice" on page 345.)
"edit"	edit, text	This creates a command button that allows the user to enter text. For each character typed into the edit area, the command is sent with the current text as the command's parameter.

Example

```
<command name="cmdviewashtml" style="icon" visible="true">
    <caption localeRef="btnTxtVAHtml">View As HTML      </caption>
    <toolTipText localeRef="btnVAHtml">View As HTML      </toolTipText>
</command>
```

cmd

An abbreviated version of command, created to reduce the time required to load configuration data. Note that `<cmd>` has fewer attributes than `<command>`.

The `<command>` element is also available. You must use it for more complex commands, such as dropdown lists.

Element Hierarchy

```
<config>
    <features>
        <standard> or any other feature
        <cmd>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the command enabled? If false, the command is not created.
key	String		An image internally available to the editor that appears if the command is assigned to a button. <i>See Also:</i> “Button Images” on page 360
name	String	“”	The command’s name. The name must be unique.
ref	String	“”	Replaces caption and toolTipText. Enter a code from the localization file to define a command’s caption and ToolTipText. <i>See Also:</i> “Locale Files” on page 266
src	String		An image specified by a URL (in other words, the image exists somewhere in the Internet or an Intranet). The image appears if the command is assigned to a button. <i>See Also:</i> “Button Images” on page 360
style	String		The style of the command when it appears on an icon bar or menu. <ul style="list-style-type: none"> ● “icon” ● “toggle” ● “listbox” ● “edit” For more information, see “Command Styles” on page 338

Example

```
<cmd name="cmdprint" key="print" ref="sPrint" />
```

config

The single root element that signifies that this configuration belongs to **eWebEditPro+XML**. This entry must exist before the configuration information is processed.

See Also: “The Config Element” on page 326

Element Hierarchy

<config>

Child Elements

interface, features

Attributes

Name	Attribute Type	Default	Description
product	String	“ “	The configuration data's target product. This attribute's value must be eWebEditPro for processing to continue.
version	Integer	0	The product release for which this configuration data is targeted. The value must be 2 or greater for processing to continue.
revision	Integer	0	The revision of the target product.

Example

```
<config product="eWebEditPro" version="4" revision="1">
```

features

One of the two major sections of the configuration data, the features section defines all standard and custom commands, and publishing options.

All features loaded into the product must be defined within this element. Any feature defined outside is ignored.

See Also: “The Features Element” on page 327

Element Hierarchy

<config>
 <features>

Child Elements

clean, custom tag, edithtml, external, form elements, mediafiles, mwsord, spellcheck, standard, table, viewas

NOTE

The features element has other child elements that do not affect the user interface. They are depicted in “Configuration Data: Hierarchical View” on page 322.

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	<p>If set to false, all features are disabled and no commands are created in the interface.</p> <p>If false, the client or script must use the Toolbar object to create any necessary commands.</p> <p>Note that the standard feature cannot be disabled. For this feature only, the enabled attribute is ignored.</p>

Example

```
<features enabled="true">
```

image

Specifies an image to display for a [command](#). If the command style is [toolbar button](#), the image appears on the popup or menu button.

See Also: “[Button Images](#)” on page 360, and “[Changing the Image that Appears on a Toolbar Button](#)” on page 242

Element Hierarchy

```
<config>
  <features>
    <standard>
      <command>
        <image>
```

Attributes

Name	Attribute Type	Default	Description
key	String	default	The name of the internal image to display. If an image has both a key and a src value, the src value overrides the key value.
src	String	“ “	The location of an external image. Since this is seen as a path, the character case is preserved.

Example

Using only a Key Attribute

```
<command name="cmdCut" style="icon" visible="true">
  <image key="Cut" />
  <caption localeRef="cmdCut">Cut</caption>
  <toolTipText localeRef="cmdCut">Cut a selection</toolTipText>
</command>
```

Using Both a Key and an Src Attribute

```
<command name="mysaveaspif" style="icon"           visible="true">
  <!--The src attribute takes precedence over the key attribute -->
  <image key="spellcheck" /
  src="http://us.al.yimg.com/us.yimg.com/i/ww/gift1.gif"/>
  <toolTipText localeRef="btntsapf">Save as PIF</toolTipText>
</command>
```

interface

The section of the configuration data that defines toolbars, [menus](#), dialogs, and other interface items. Interface items defined outside this section are ignored.

See Also: “[The Interface Element](#)” on page 326

Element Hierarchy

```
<config>
  <interface>
```

Child Elements

[menu](#), [popup](#)

Attributes

Name	Attribute Type	Default	Description
allowCustomize	Boolean	Yes	<p>Determines whether users can customize their interface from the one defined in the configuration data.</p> <p>If True, the user can modify toolbars. The customization is saved on the local system under the name given in the <code>name</code> attribute.</p> <p>If you set this value to False, the editor ignores any customization that the user saves. In this case, the default interface is used.</p> <p><i>See Also:</i> "Letting Users Customize the Toolbar" on page 315</p>
enabled	Boolean	Yes	<p>Determines whether the interfaces defined here are enabled. If set to False, it is the responsibility of the client or script to use the <code>Toolbar</code> object to create the interface.</p> <p><i>See Also:</i> "Dynamically Changing the Editor" on page 250</p>
name	String	Default	<p>The name of the interface. When a user customizes their interface, this name identifies the changes.</p> <p>One method of resetting an interface to allow for customization, but ignore previous customization, is to change the name. This will ignore a saved configuration and use the one defined in the configuration data.</p> <p><i>See Also:</i> "Letting Users Customize the Toolbar" on page 315</p>

Name	Attribute Type	Default	Description
visible	Boolean	True	<p>Controls whether the toolbar is visible. If set to false, the interface is created but does not appear. However, the context menu appears if a user right clicks the mouse.</p> <p>If set to false, the toolbar can only be displayed by programmatically by calling <code>ShowAllMenus()</code> in the Menus interface, using a script like this:</p> <pre>eWebEditPro.instances.MyContent1.editor.Menus().ShowAllMenus();</pre> <p>(See Also: “Method: ShowAllMenus” on page 127.)</p> <p>For example, you set this attribute to false because the editor is the second one on a page. The XML data would look like this:</p> <pre><interface name="standard1" allowCustomize="true" visible="false"></pre> <p>But, if the user's focus shifts to the second editor, you want to display its toolbar. At that point, you display the toolbar using this script:</p> <pre>eWebEditPro.MyContent2.Menus().ShowAllMenus();</pre>
context	Boolean	True	<p>Controls whether the context menu is visible. If set to true, a menu appears when the user right clicks the mouse with choices that are unique to the current situation (or context).</p> <p>For example, if you are editing text and right click the mouse, the context menu displays common editing commands, such as cut and copy text. If you are editing a table, the context menu displays commands relevant to that activity, such as insert row and insert column.</p> <p>If set to false, the context menu does not appear.</p>

Example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="4" revision="1">
  <interface name="myinterface" allowCustomize="true">
```

listchoice

Defines an individual choice in a list box command item.



Use this element to define attributes for each item in a list. The listchoice command's style attribute must be set to **List** or **Listbox**.

See Also: “Determining which Fonts, Font Sizes, and Headings are Available” on page 246.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <command>
        <selections>
          <listchoice>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the item enabled? If not, it is excluded from the list.
command	String	“ “	The command to send in place of the command that contains the list. If not specified or empty, the listchoice command is sent, and the selection's index or assigned data is sent as parameters.
data	Integer	0	This value is assigned to the item selection. It is sent with the command as a parameter. If the value assigned is zero (0), the index of the selection is sent.
localeRef	String	“ “	Used to translate the #text attribute of the element. See Also: “Translating Button Captions and Tool Tips” on page 244
#text	String	“ “	The command checks the body of the listchoice element to see if it includes text. <ul style="list-style-type: none"> • If it does, that text is sent as the attribute. • If the element does not include text, the command's caption attribute is sent as the attribute. Also, this text is the selection item text. In other words, it is the list of options that the user sees on the dropdown list. This applies whether the body of the listchoice element includes text or if the command's caption attribute is used.

Example

Here is the list that creates the font choice menu shown above. Note that the items appear in the listbox in the order in which they are entered into the command.

NOTE

The top item in the list is the default value, unless the list is a font name, font size, or header style list. In that case, the currently selected item is the default value.

```

<command name="cmdfontname" style="list" visible="true" maxwidth="12">
<image key="fontname"/>
<caption localeRef="btntxtfntnm">Times New Roman, Arial </caption>
<toolTipText localeRef="bttnfntnm">Font</toolTipText>
<selections name="headinglist" enabled="true" sorted="true">
    <listchoice>Arial, Helvetica</listchoice>
    <listchoice>Comic Sans MS</listchoice>
    <listchoice>Courier New, Courier</listchoice>
    <listchoice>Symbol</listchoice>
    <listchoice>Times New Roman, Times</listchoice>
    <listchoice>Verdana, Helvetica</listchoice>
</command>

```

Using the Selections Element

You can use the **selections** element to define a group of items in a list. This can be helpful when you want to enable or disable a group of elements from one line of the configuration data.

Parameters to the Listchoice Command

When a listchoice command is executed, three parameter values are sent along with the command. Note that all commands can include a name and a text parameter.

Parameter	How Value Determined
name	The command checks to see if a command attribute is assigned to the listchoice element. <ul style="list-style-type: none"> If a command attribute is assigned to the element, the system sends that command. If a command attribute is not assigned to the element, the system sends the higher level command to which the listchoice command is assigned.
text	The command checks the body of the listchoice element to see if it includes text. <ul style="list-style-type: none"> If it does, that text is sent as the parameter. If the item does not include text, the defined command's caption attribute is sent as the parameter.
data	The data value assigned to the item selection. If the value assigned is zero (0) (the default value), the index of the selection is sent.

Assigning Command Attributes to Listchoice Elements

If you wish to send a list item as a **command** rather than parameter data, place a command attribute in each listchoice element.

Commands assigned as attributes to listchoice elements do not need to be defined as other commands are (that is, under the commands section of the configuration data). If a command is defined under the commands section of the

configuration data, information about that command (such as the caption) is used with the selection.

Not Assigning Command Attributes to Listchoice Elements

If no command attributes are assigned to a listchoice element, the command that contains the list is sent instead, and the index or data of the selection is sent as a parameter.

Example

```
<command name="mylist" style="list" visible="true">
<image key="Spelling"/>
<caption localeRef="btnTxtsapf">Dropdown List of Commands</caption>
<toolTipText localeRef="btntsapf">Dropdown List of Commands </toolTipText>
<selections name="testlist" enabled="true" sorted="true">
    <listchoice command="cmdcopy" data="50" enabled="true"></listchoice>
    <listchoice command="cmdcut" data="8"></listchoice>
    <listchoice command="cmdpaste" data="107" localeRef="cmpaste">Just Paste</listchoice>
    <listchoice command="mynotify" data="42" localeRef="mynotify">Undefined Command </listchoice>
    <!--These selections generate the larger command name "mylist" -->
    <listchoice data="99" enabled="false">Do Not Show This</listchoice>
    <listchoice data="999" enabled="true">Selection B</listchoice>
</selections>
</command>
```

menu

Defines a toolbar or pulldown menu.

A menu is the interface between the user and the [commands](#).

See Also: “[Creating a Popup Menu](#)” on page 245 and “[Determining Which Menus Appear on the Toolbar](#)” on page 231.

Element Hierarchy

```
<config>
    <interface>
        <menu>
```

Child Elements

[caption](#), [bar](#), [button](#), [space](#)

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	Is the menu enabled? A false value prevents the creation of the menu.
name	String	“ “	The menu name. You refer to a menu by its name. The name must be unique.
newrow	Boolean	Yes	If the menu style is icon bar, a yes value forces the menu to a new line on the toolbar. See Also: "Placing a Toolbar Menu on a Row with Another Menu" on page 235 .
showbuttonscaptions	Boolean	False	If true, button captions are shown. Otherwise, tool tips act as the caption. This is one of the few attributes that defaults to false. See Also: "Creating or Editing the Toolbar Menu Caption" on page 236 and "Translating Button Captions and Tool Tips" on page 244 .
style	String	icon	Defines the look of the menu. These are the styles. “Icon” (default) - Toolbar “Pulldown” - Dropdown list “Tab” - Tab Selections “Status” - Status bar “Popup” - Context Menu (See Also: "popup" on page 351)
textalignment	String	Yes	Alignment of the text on the button. (Only used if showbuttonscaptions is set to “true” .) These are the valid values. “Top” “Left” “Right” “Bottom” “Center” The default value is Bottom . See Also: "Defining the Alignment of Caption Text" on page 243 .

Name	Attribute Type	Default	Description
visible	Boolean	True	<p>Determines whether the menu appears within the editor by default. If set to false, the user must perform an action to display the menu.</p> <p>For example, the user may have to select the menu from a dropdown list to have it appear on the toolbar.</p>
wrap	Boolean	True	<p>If true, and a toolbar, when the icons reach the right edge of the display area, they wrap to the next line.</p> <p>If false, the icons do not wrap to the next line. They are invisible until you move the menu bar to another line of the toolbar.</p> <p>See Also: "Determining if a Toolbar Menu Should Wrap to the Next Row" on page 235.</p>

Example

```
<menu name="editbar" newRow="true"
      showButtonsCaptions="false" textAlign="bottom">
  <caption visible="false" localeRef="btnMainCap">Edit </caption>
  <button command="cmdcut" />
  <button command="cmdcopy" />
  <button command="cmdpaste" />
</menu>
```

popup

Defines a popup menu. This menu is pre-defined for use either as a stand-alone menu that is invoked programmatically, or as a menu attached to a **command** button.

For more information, see ["Creating a Popup Menu" on page 245](#).

Element Hierarchy

```
<config>
  <interface>
    <popup>
```

Child Elements

[caption](#), [bar](#), [button](#), [space](#)

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the menu enabled? If set to false, you cannot create this menu.
name	String	""	The menu name. You refer to a menu by its name. The name must be unique.

Example

```
<popup name="ViewAsPopup"
      <caption visible="0" localeRef="btnMyViewAs">View As</caption>
      <button command="cmdviewaswysiwyg" />
      <button command="cmdviewashtml" />
    </popup>
```

selections

Defines a list of items within a `listchoice` command.

This element can be helpful when you want to enable or disable a group of elements from one place.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <command>
        <selections>
```

Child Elements

`listchoice`

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the list enabled? If no, then the defined list is ignored.
name	String	""	The name of the list. It must be unique.
sorted	Boolean	True	If set to <ul style="list-style-type: none"> • “true”, the list appears in alphabetical order • “false”, the list appears as entered in the configuration data

Example

```
<selections name="testlist" enabled="true" sorted="true">
    <listchoice command="cmdcopy" data="50" enabled="true"></listchoice>
    <listchoice command="cmdcut" data="8"> </listchoice>
    <listchoice data="999" enabled="true">Send testlist Command
</listchoice>
</selections>
```

space

Places a blank separator between toolbar buttons or [popup](#) menus. On a toolbar, a space is one half the width of a normal icon (8 pixels).

The space command makes the toolbar easier to read.

NOTE Unlike the other commands, the bar and space elements are not defined in the configuration data. You cannot modify their appearance.

Buttons with a Space Command



Buttons without a Space Command



See Also: “[Adding a Space Between Two Toolbar Menu Items](#)” on page 241

Element Hierarchy

```
<config>
    <interface>
        <menu>
            <space>
```

```
<config>
  <interface>
    <popup>
      <space>
```

Attributes

Name	Attribute Type	Default	Description
None			There are no attributes to the space element.

Example

```
<menu name="editbar">
  <caption visible="false" localeRef="btnMainCap">Edit</caption>
  <button command="cmdcut" />
  <button command="cmdcopy" />
  <space/>
  <button command="cmdpaste" />
</menu>
```

standard

Defines standard editing commands and options.

Element Hierarchy

```
<config>
  <features>
    <standard>
```

Child Elements

command, cmd, style

Attributes

Name	Attribute Type	Default	Description
autoclean	Boolean	"True"	<p>Whether the editor automatically detects content created by Microsoft Office 2000 applications (for example, Word 2000). Office 2000 content may cause problems when eWebEditPro+XML users try to reformat it (for example, change the font size).</p> <p>"false" - Do not detect Office 2000 content "true" (default) - Detect Office 2000 content</p> <p>When the editor detects this content, the <code>prompt</code> attribute of the <code><clean></code> element determines if a message appears, asking the user whether or not to clean the HTML code. (Answering yes to the prompt is the same as selecting Clean HTML from the right-click menu.)</p> <p>See Also: "prompt" on page 398</p>
publish	String	"xhtml"	<p>Allows you to determine whether editor content is stored as HTML or XHTML.</p> <hr/> <p>Important: If you are using eWebEditPro+XML with an Ektron CMS, leave this setting as xhtml.</p> <hr/> <p>"xhtml" - The HTML code is converted to the XHTML 1.0 standard (as defined at http://www.w3.org/TR/xhtml1). "Contumely" - Medium level: eliminates overlapping tags, and merges font tags. "Minimal" - Eliminates invalid fonts, filters image urls, and replaces cr/lf according to the values set in those attributes.</p> <p>Advantages of XHTML XHTML can be parsed by an XML parser. Also, XHTML has replaced HTML as a W3C recommendation.</p> <p>Advantages of HTML HTML is more likely to be compatible with older versions of browsers. If you are unfamiliar with XHTML, choose HTML.</p>

Name	Attribute Type	Default	Description
shiftenter	Boolean	“false”	<p>By default, eWebEditPro+XML inserts a paragraph tag (<p>) when the user presses <Enter>.</p> <p>To change this behavior so that a linebreak
 appears when the user presses <Enter>, set it to true.</p>
publishviewassource	Boolean	“true”	<p>Prevents content from being saved in “View As HTML” mode. If <code>publishviewassource="false"</code>, the editor switches to WYSIWYG mode when the user saves the content. This lets the user review the content ‘s format before saving.</p> <p>The default value, “true”, allows the content to be saved “as is” when in View As HTML mode.</p>
default div on enter	Boolean	“false”	<p>If set to true, a <DIV> tag is inserted when a user presses <Enter>, instead of a <P> tag. (This only occurs if there is no preceding <P> tag.)</p> <p>By default, a <DIV> tag has single spacing between paragraphs, while <P> tags have double spacing, unless otherwise specified in a style sheet.</p> <p>See Also: “shiftenter” on page 356</p> <p>If <code>shiftenter</code> is set to true, this attribute is ignored.</p> <p>Also, if a user presses <Enter> after a <DIV> tag, another <DIV> tag is inserted, regardless of how this attribute is set. This is the browser’s default behavior.</p>
continueparagraph	Boolean	“false”	<p>If set to true, removes the leading <P> (or <DIV>) tag and its corresponding closing tag. This is useful when the content will be appended to an existing paragraph, so you do not want to start a new paragraph.</p>
maxloadsec	Number	20	<p>Determines the number of seconds to wait for a document to load before displaying a message that the document is taking too long.</p> <p>See Also: “docbusymsg” on page 357</p> <p>This message asks the user if he or she wants to wait or to proceed as if the loading were done. The delay is typically caused by trying to resolve non-existent links, but can also be caused when many editors are on a page and share document processing time.</p> <p>This attribute lets the developer control how long to wait before showing the warning. For example, if you know that the page has many editors that take a long time to load, increase this value to increase the time that elapses before the warning appears.</p>

Name	Attribute Type	Default	Description
docbusymsg	Boolean	“true”	<p>Specifies whether to display the Document is busy dialog. This dialog appears when eWebEditPro+XML is busy resolving an address, processing document objects, etc. and cannot fully load a document. The waiting time is set via the <code>maxloadsec</code> attribute.</p> <p><i>See Also:</i> “maxloadsec” on page 356</p> <p>If this value is “true” and the wait period elapses, the “Document busy” dialog appears. The user can wait for the document to load or continue without waiting.</p> <p>If the value is false and the wait period elapses, the dialog does <i>not</i> appear, and the document does not load into eWebEditPro+XML. The editor assumes the user does <i>not</i> want to wait for the document to complete loading.</p> <hr/> <p style="color: red;">Warning: If the document has not completed processing, the retrieved document may be empty. So, you should only set a false value when you know the content can be successfully retrieved. Instead of using this attribute, it's better to use the <code>maxloadsec</code> attribute to increase the wait time.</p> <hr/>

style

Defines style sheet and controls other aspects of style sheet implementation.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <style>
```

Attributes

Name	Attribute Type	Default	Description
publishstyles	Boolean	“false”	Determines whether style sheet specifications for each tag are inserted into file when the content is saved. <i>See Also:</i> “Saving Style Sheet Tags When Content is Saved” on page 434
href	string	[eWebEditPro Path]/ektnormal.css	Sets the location of the style sheet. <i>See Also:</i> “The Default Style Sheet” on page 431
preserveword styles	Boolean	“true”	Determines whether Word style attributes (those with so- in them) are preserved when Microsoft Office 2000 or later content is pasted into the editor. <i>See Also:</i> “Preserving Tags When Office Content is Pasted” on page 433
preserveword classes	Boolean	“true”	Determines whether Word class tags are preserved when Microsoft Office 2000 or later content is pasted into the editor. <i>See Also:</i> “Preserving Tags When Office Content is Pasted” on page 433
wrapstylewith div	Boolean	“true”	Determines what to do when a user applies a generic style class to text surrounded by blocking tags. <i>See Also:</i> “Inserting span or div Tags” on page 434

Example

```
<style publishstyles="false" href="[eWebEditProPath]/ektnormal.css" equivClass="strict"
wrapstylewithdiv="false" preservewordstyles="false" preservewordclasses="true">
</style>
```

toolTipText

Defines the tool tip text that pops up when the cursor hovers over an icon.



Element Hierarchy

```
<config>
  <features>
```

```
<standard>
<command>
    <toolTipText>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	Yes	Is the element enabled?
localeRef	String	""	Used as an identifier to translate the element's #text attribute. See Also: “Translating Button Captions and Tool Tips” on page 244
text	String	""	The text in the tool tip.

Example

```
<command name="cmdLeft" style="2" visible="true">
    <image key="Left" />
    <caption localeRef="btnTxtal">Left</caption>
    <toolTipText localeRef="bttnal">Align Left</toolTipText>
</command>
```

Button Images

Images are available to be placed on [buttons](#). Assign an image element to a [command](#) to specify the image appears when the command is selected.

See Also:

- “[Changing the Image that Appears on a Toolbar Button](#)” on page 242
- “[image](#)” on page 342
- “[Images Supplied by eWebEditPro+XML](#)” on page 360

Formats Supported

eWebEditPro+XML supports the following image formats.

- Windows Bitmap
- GIF
- JPEG

Sources of Images

There are two sources of images, and two kinds of image command elements.

- **Images supplied by eWebEditPro+XML** - Specify these by entering the image command's `key` attribute.

For example: `<image key="cut" />`

In this example, `cut` is the keyword that specifies the image. For a list of standard image keywords and associated images, see “[Images Supplied by eWebEditPro+XML](#)” on page 360.

- **Images from another source**, such as those created by your organization - specify these by entering a URL using the image command's `src` attribute. The URL can refer to a local or remote location.

For example:

```
<image src="http://www.yourcompany.com/images/mycut.gif" />
```

If an image has a `key` and an `src` value, the `src` attribute overrides the `key`.

For more information, see “[Creating Your Own Images](#)” on page 369.

Images Supplied by eWebEditPro+XML

The table on the following page lists the image keywords and associated images supplied by eWebEditPro+XML. Some images are only available if

your organization has purchased WebImageFX. These are indicated by an asterisk(*)).

NOTE

Note that eWebEditPro+XML also supplies a set of special characters that can appear on toolbar buttons. See "Special Character Commands" on page 212.

- about - 
- abovetext - 
- absmode - 
- abspos - 
- additem - 
- alert - 
- audio - 
- back - 
- backward - 
- balloon - 
- bar - 
- bbtn - 
- belowtext - 
- bgcolor - 
- blank - 
- *blur - 
- bold - 
- bookmark - 
- books1 - 
- books2 - 
- books3 - 
- borders - 

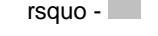
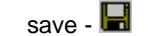
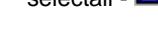
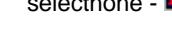
- borders2 - 
- *brightness - 
- browse - 
- bull - 
- bullets - 
- camera - 
- cellprop - 
- center - 
- charsmenu - 
- check1 - 
- checkbox - 
- choice - 
- clean - 
- close - 
- *colordepth - 
- comment - 
- *contrast - 
- copy - 
- *crop - 
- cut - 
- dagger - 
- ddagger - 
- del - 
- delete - 
- delrow - 

- details - 
- default - 
- delcell - 
- delcol - 
- *dimensions - 
- *digitalcamera - 
- dnload - 
- droplist - 
- erase - 
- *eraser - 
- euro - 
- eyeglasses - 
- faceplain - 
- facesad - 
- facesmile - 
- fgcolor - 
- fileup - 
- find - 
- findnext - 
- *floodfill - 
- floppy - 
- fnof - 
- fontcolor - 
- fontcolor2 - 
- fontcolpal - 
- fontface - 

- fontsize - 
- form - 
- forward - 
- freehand - 
- front - 
- glyphs - 
- hellip - 
- help - 
- helpwhat - 
- hiddenfld - 
- hiliter - 
- hiliter1 - 
- hiliter2 - 
- *horizflip - 
- horzrule - 
- hyperlink - 
- hyperlinkstar - 
- indentleft - 
- indentright - 
- info - 
- *information - 
- ins - 
- inscell - 
- inscol - 
- insrow - 
- instable - 

- italic - 
- justify - 
- key - 
- ldquo - 
- ldquor - 
- left - 
- *line - 
- link - 
- lock - 
- lsquo - 
- lsquo - 
- lsquor - 
- ltrblock - 
- ltredit - 
- mail - 
- math - 
- mdash - 
- mergecell - 
- msword - 
- ndash - 
- new - 
- newwin - 
- nojustify - 
- *normalview - 
- note - 

- numbered - 
- oelig - 
- oeoelig - 
- one - 
- open - 
- optionbox - 
- *oval - 
- page - 
- pagetag - 
- *palette - 
- paperclip - 
- paste - 
- *pastenew - 
- pastetext - 
- pencil - 
- pencil1 - 
- pencil2 - 
- permil - 
- picture - 
- plain - 
- *polygon - 
- preview - 
- print - 
- properties - 
- pwdfld - 
- question - 

- rbtn - 
- rdquo - 
- *rectangle - 
- redo - 
- removelink - 
- removestyle - 
- right - 
- *rotate - 
- rsaquo - 
- rsquo - 
- rtlblock - 
- rtledit - 
- save - 
- saveall - 
- *saveas - 
- sbtn - 
- scaron - 
- *select - 
- selectall - 
- selectnone - 
- setup - 
- snapgrid - 
- space - 
- spellayt - 
- spellcheck - 

- splitcell - 
- *spraycan - 
- sscaron - 
- strikethrough - 
- subscript - 
- superscript - 
- table - 
- tablemenu - 
- tableprop - 
- table508 - 
- text - 
- textbox - 
- textfld - 
- three - 
- thumbnail - 
- timer - 
- trade - 
- *twain - 
- two - 
- underline - 
- undo - 
- up1lv - 
- *update - 
- upload - 
- *vertflip - 

- vidcam - 
- viewprop - 
- wand - 
- warning - 
- world - 
- world2 - 
- yyuml - 
- zcaron - 
- zoomin - 
- zoomout - 
- zordermenu - 
- zzcaron - 

Creating Your Own Images

You can create your own custom button images for the **eWebEditPro+XML** toolbar.

To create a new button image or modify an existing one, you can use any commercially available paint program that can produce GIF files.

By convention, button image file names start with "btn".

See Also: ["Changing the Image that Appears on a Toolbar Button" on page 242](#)

Image File Extensions

Although the graphic file for a toolbar button is usually a GIF (.gif) file, it can also be a Windows bitmap (.bmp), or a JPEG (.jpg) file. Windows bitmap files are larger than GIF and, therefore, take longer to download. JPEG files are optimized for photographs and images and usually do not display a small icon clearly. As a result, the GIF file format is preferred.

Size of Button Images

Although a button image can be almost any size, the standard size provided with **eWebEditPro+XML** is 16 by 16 pixels. If you wish, you could create buttons of a larger uniform size, as is common with Microsoft Internet Explorer, but the **eWebEditPro+XML** toolbar would occupy more space on your Web page.

Background Color of Button Images

Also, a button image's background color should conform to the Windows' background color for buttons and other 3D objects. Any pixel that is gray (hex value C0C0C0) will display as the Windows' button (3D Objects) color.

Button Image Specification Summary

Image Attribute	Value	Comments
File Format	GIF	JPEG (JPG) and Windows Bitmap (BMP) also supported.
Width	16 pixels	Any size is possible; this is the standard size.
Height	16 pixels	Any size is possible; this is the standard size.
Background Color	RGB: 192, 192, 192; Hex: C0C0C0	Other colors do not conform to the Windows' background color.
File name prefix	btn	The prefix is only a convention, not a requirement.

Managing Tables

eWebEditPro+XML's configuration data lets you determine whether users can enter tables into the editor. If you decide that they can, you can restrict the list of commands that users can perform on them. For example, you may decide that users cannot add or remove columns. You can also customize the tables menu and the tables toolbar menu.

NOTE

You cannot customize the context-sensitive menu.

In addition, you can specify default values for the Insert Table dialog box, and control the responses that users can enter into the **Horizontal Alignment** and **Vertical Alignment** fields of the Table and Cell properties dialog boxes.

This section explains

- whether users can **enter tables** into the editor
- how to **customize** the table and cell property dialogs
- how to **restrict the list** of table options
- how to customize the options on the
 - **tables** menu
 - **table toolbar** menu
- **setting default values** for the Insert Table dialog box
- **controlling the responses** for the **Horizontal Alignment** and **Vertical Alignment** fields

The Table Element of the Configuration Data

Defines options that appear on table menus.

Element Hierarchy

```
<config>
  <features>
    <table>
```

Child Elements

`cmd`

Attributes

`none`

Allowing Users to Create Tables

To allow users to create tables in the editor, set the `enabled` attribute of the `table` command to **true**. If you set `enabled` to **false**, the Insert Tables button () and the table menus do not appear.

Below is the section of the configuration data that enables users to create tables.

See Also: “Table Commands” on page 207

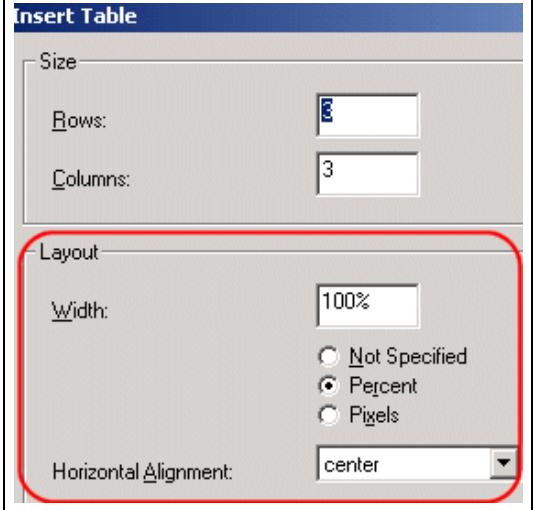
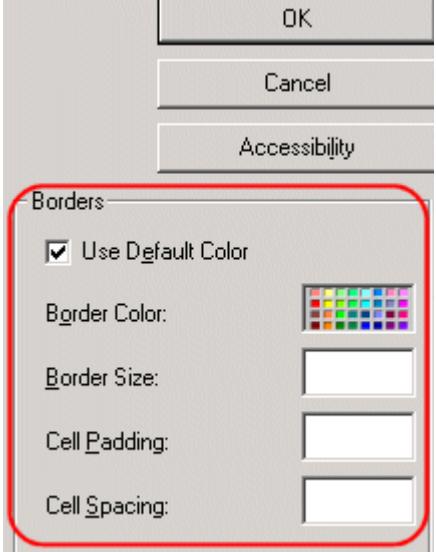
```
<table enabled="true"
visiblelayoutframe="true"
visibleborderframe="true"
visiblebackgroundframe="true"
visibleaccessibilitybtn="true"
visiblecellaccessibilityframe="true"
<cmd name="cmdtable" key="tablemenu" ref="mnuTbl" />
<cmd name="cmdinserttable" key="instable" ref="mnuITbl" />
<cmd name="cmdinsertrow" key="insrow" ref="mnuIRow" />
<cmd name="cmdinsertcolumn" key="inscol" ref="mnuICol" />
<cmd name="cmdinsertcell" key="inscell" ref="mnuICell" />
<cmd name="cmddeleterows" key="delrow" ref="mnuDRow" />
<cmd name="cmddeletecolumns" key="delcol" ref="mnuDCol" />
<cmd name="cmddeletecells" key="delcell" ref="mnuDCell" />
<cmd name="cmdmergecells" key="mergecell" ref="mnuMC" />
<cmd name="cmdsplitsplitcell" key="splitcell" ref="mnuSC" />
<cmd name="cmdtableproperties" key="tableprop" ref="mnuTProp" />
<cmd name="cmdcellproperties" key="cellprop" ref="mnuCProp" />
</table>
```

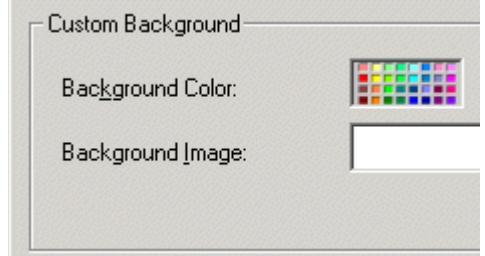
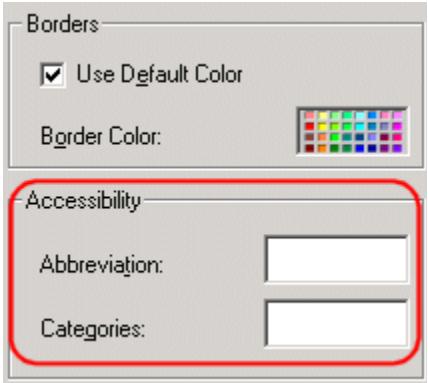
Customizing the Table Dialogs

Within the `<table>` element of the configuration data, five properties let you control the appearance of the Insert Table and Cell Properties dialogs.

```
<table enabled="true"
visiblelayoutframe="true"
visibleborderframe="true"
visiblebackgroundframe="true"
visibleaccessibilitybtn="true"
visiblecellaccessibilityframe="true"
```

They are described below.

Attribute	Determines whether this section of the screen appears	Example of screen section
visiblelayoutframe	Layout area of Insert Table dialog	 <p>The screenshot shows the 'Insert Table' dialog with the 'Layout' section selected. The 'Width:' field contains '100%' and the 'Horizontal Alignment:' dropdown is set to 'center'. A red box highlights this entire section.</p>
visibleborderframe	Borders area of Insert Table dialog	 <p>The screenshot shows the 'Insert Table' dialog with the 'Borders' section selected. It includes options for 'Use Default Color' (checked), 'Border Color:' (color palette), 'Border Size:' (input field), 'Cell Padding:' (input field), and 'Cell Spacing:' (input field). A red box highlights this entire section.</p>

Attribute	Determines whether this section of the screen appears	Example of screen section
visiblebackgroundframe	Custom background area of Insert Table dialog	
visibleaccessibilitybtn	Accessibility button of Insert Table dialog	
visiblecellaccessibilityframe	Accessibility area of Cell Properties screen	

Restricting Table Options

If you want to let users insert tables but determine which commands users can perform on them, remove unwanted commands from between the <table> tags of the XML configuration data. (To learn about table commands, see “[Table Commands](#)” on page 207.)

For example, to remove the Insert Row and Delete Rows commands, delete the two lines indicated by strikethrough below.

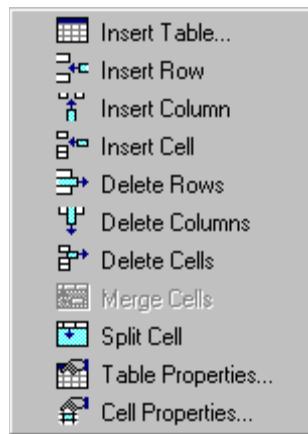
```
<table enabled="true">||
<cmd name="cmdtable" key="tablemenu" ref="cmdTbl"/>||<cmd name="cmdinserttable" key="instable" ref="mnuiTb1"/>||
<cmd name="cmdinsertrow" key="insrow" ref="mnuiRow"/>||
<cmd name="cmdinsertcolumn" key="inscol" ref="mnuiCol"/>||<cmd name="cmdinsertcell" key="inscell" ref="mnuiCell"/>||
<cmd name="cmddeleterow" key="delrow" ref="mnuiBRow"/>||
<cmd name="cmddeletecolumns" key="delcol" ref="mnuiCol"/>||<cmd name="cmddeletecells" key="delcell" ref="mnuiCell"/>||
<cmd name="cmdmergecells" key="mergecell" ref="mnuiMC"/>||<cmd name="cmdsplitchell" key="splitcell" ref="mnuiSC"/>||
<cmd name="cmdtableproperties" key="tableprop" ref="mnuiTProp"/>||<cmd name="cmdcellproperties" key="cellprop" ref="mnuiCProp"/>||
</table>||
```

Commands removed from this list do not appear on menus that list table options.

Customizing the Tables Menu

If the cursor is within a table when a user clicks the Insert Tables button (

), a menu of options appears (illustrated below).



NOTE

Commands that cannot be performed are “grayed out.” For example, because only one cell is selected, the Merge Cells option is grayed out.

To restrict the options on this menu, edit the list of commands between the `<popup name="tablepopup">` tags in the configuration data. (To learn about the table commands, see [“Table Commands” on page 207](#).)

For example, to remove the Insert Row and Delete Rows commands, delete the two lines indicated by strikethrough below.

```
<popup name="tablepopup">
<caption localeRef="cmdTbl" />
```

```

<button command="cmdinserttable" />
<button command="cmdinsertrow" />
<button command="cmdinsertcolumn" />
<button command="cmdinsertcell" />
<button command="cmddeleterows" />
<button command="cmddeletecolumns" />
<button command="cmddeletecells" />
<button command="cmdmergecells" />
<button command="cmdsplitscell" />
<button command="cmdtableproperties" />
<button command="cmdcellproperties" />-->
</popup>

```

Customizing the Tables Toolbar Menu

The tables toolbar menu appears if the user adds it to the toolbar or the menu's `visible` attribute is set to **true** in the configuration data. (To learn how users add to the toolbar, see "Customizing Your Toolbar" in the **eWebEditPro+XML** User Guide.)

If the user displays the tables toolbar menu, its default appearance is below.



NOTE

Commands that cannot be performed are "grayed out." For example, because only one cell is selected, the **Merge Cells** option is grayed out.

To restrict the options on this menu, edit the list of commands between the `<menu name="toolbar" ...>` tags in the configuration data. (To learn about the table commands, see "Table Commands" on page 207.)

For example, to remove the Insert Row and Delete Rows commands, delete the two lines indicated by strikethrough below.

```

<menu name="toolbar" newRow="true" showButtonsCaptions="false"
wrap="false" visible="false">
    <caption localeRef="cmdTbl" />
    <button command="cmdinserttable" />
    <button command="cmdinsertrow" />
    <button command="cmdinsertcolumn" />
    <button command="cmdinsertcell" />
    <button command="cmddeleterows" />
    <button command="cmddeletecolumns" />
    <button command="cmddeletecells" />
    <button command="cmdmergecells" />
    <button command="cmdsplitscell" />
    <button command="cmdtableproperties" />
    <button command="cmdcellproperties" />
</menu>

```

Setting Default Values for the Insert Table Dialog

You can customize the default values that appear in the Insert Table dialog box. To do this, enter a text data argument of HTML table attributes when sending the command in JavaScript.

For example:

```
if ("jsinstable" == strCmdName)

var strAttrs = "rows=6 cols=3 width='75%' bgcolor='cyan'
border=2 borderColor=navy cellpadding=2 cellspacing=3 rules=cols";
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdinserttable", strAttrs, 0);
return(true);
}
```

In this example, the Insert Table dialog box is launched when the user presses a custom button whose command is `jsinstable`.

The table dialog box will appear with default values specified in the attributes string. The number of rows and columns can be specified using the pseudo attributes 'rows' and 'cols' respectively. You can also specify attributes that do not appear in the dialog, such as `rules="cols"` in the above example.

The following table explains how to set a default value for each field in the Insert Table dialog.

Field	How to Set Default Value
Rows	<code>rows=number of rows</code>
Columns	<code>cols=number of columns</code>
Width	<code>width=number of pixels or percentage</code>
Horizontal Alignment	See “Controlling Alignment Field Responses” on page 378
Border Color	<code>borderColor=color name or hexadecimal code</code>
Border Size	<code>border=number of pixels</code>
Cell Padding	<code>cellpadding=number of pixels</code>
Cell Spacing	<code>cellspacing=number of pixels</code>
Background Color	<code>bgcolor=color name or hexadecimal code</code>

Field	How to Set Default Value
Background Image	background= <i>url of image</i>

Entering the Sample Code

Enter the sample code in a customevents.js file, in a onexeccommand handler function (for details, see “[Creating a Custom Command](#)” on page 215). The command is executed when the user selects it from a custom dropdown list or presses a custom button.

To learn how to create a custom dropdown list, see “[Creating a Popup Menu](#)” on page 245.

To learn how to create a custom button, see “[Creating a Custom Command](#)” on page 215.

Controlling Alignment Field Responses

In the configuration data, you can determine the possible responses and a default response for the following fields.

Dialog Box	Fields
Table Properties	Horizontal Alignment
Cell Properties	Horizontal Alignment, Vertical Alignment

NOTE You can only enter one set of responses for both Horizontal Alignment fields. In other words, you cannot specify one set of responses for the Horizontal Alignment field in the Table Properties box and a different set for the Horizontal Alignment field in the Cell Properties box.

Controlling Responses for the Horizontal Alignment Field

To specify the list of responses for the **Horizontal Alignment** field, add the following code between the <table> tags in the configuration data. In this example, **center** is designated as the default response, because it has the `default="true"` attribute.

```
<selections name="align">
    <listchoice value="left" localeRef="tblHAL"/>
    <listchoice value="center" localeRef="tblHAC" default="true"/>
    <listchoice value="right" localeRef="tblHAR"/>
```

```
<listchoice value="justify" localeRef="tblHAJ"/>
<listchoice/>
</selections>
```

Controlling Responses for the Vertical Alignment Field

To specify the list of responses for the **Vertical Alignment** field, add the following code between the `<table>` tags in the configuration data. In this example, **middle** is designated as the default response, because it has the `default="true"` attribute.

```
<selections name="valign">
  <listchoice/>
  <listchoice value="top" localeRef="tblVAT"/>
  <listchoice value="middle" localeRef="tblVAM" default="true"/>
  <listchoice value="bottom" localeRef="tblVAB"/>
  <listchoice value="baseline" localeRef="tblVABL"/>
</selections>
```

To remove any response from the list, delete the line. To change the default, move `default="true"` to the desired alignment value.

Fonts and Headers

The font and header commands, listed below, let you specify font sizes, font styles and heading levels.

- fonts
- fontname
- fontsize
- headings
- heading[x]

See Also: “Determining which Fonts, Font Sizes, and Headings are Available” on page 246.

fonts

The section that specifies the font names and sizes that users can apply to text in the editor.

The font element provides one way to define font information. However, the preferred way is to use a **selections** element group to generate a list of fonts and commands to set them.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <fonts>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the font listing is not used.

fontname

Specifies the name of a font. The font is added to the font list made available to the user.

The name specified in the text attribute is exactly what is placed in the font tag. Be sure to use font names that browsers can interpret.

Remarks

The fontname element is one way to define a set of font names. However, the preferred method is to use a **selections** element group to define available font names.

Here is an example of the preferred method of defining font names.

```
<command name="cmdfontname" style="list" visible="true">
    <toolTipText localeRef="bttnfntnm">Font Name</toolTipText>
    <selections name="fontnamelist" enabled="true" sorted="true">
        <listchoice>Arial, Helvetica</listchoice>
        <listchoice>Courier</listchoice>
        <listchoice>Microsoft Sans Serif, Sans Serif</listchoice>
        <listchoice>Symbol</listchoice>
        <listchoice>Times New Roman</listchoice>
        <listchoice>Verdana</listchoice>
        <listchoice>Webdings</listchoice>
    </selections>
</command>
```

Element Hierarchy

```
<config>
    <features>
        <standard>
            <font names>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the font is not used.
#text	String	""	The text that the user views to select a font; also, the text placed into the font tag as the font name.

fontsize

Defines a font size that users can apply to text within the editor.

Remarks

Below are the font size commands available. These commands control the setting of the font size. They do not need to be defined in the feature as commands.

They are available for list box commands and scripting.

- cmdfontsize1
- cmdfontsize2
- cmdfontsize3

- cmdfontsize4
- cmdfontsize5
- cmdfontsize6
- cmdfontsize7

The fontsize element is one way to specify font sizes. However, the preferred way is to use the `selections` element group to generate a list of fonts and commands to set them. Here is an example use of the preferred method.

```
<command name="cmdfontsize" style="list" visible="true">
  <image key="fontsize" />
  <toolTipText localeRef="btntfontsz">Font Size</toolTipText>
  <selections name="fontsizeList" enabled="true" sorted="true">
    <listchoice command="cmdfontsize7">7 pt</listchoice>
    <listchoice command="cmdfontsize6">6 pt</listchoice>
    <listchoice command="cmdfontsize5">5 pt</listchoice>
    <listchoice command="cmdfontsize4">4 pt</listchoice>
    <listchoice command="cmdfontsize3">3 pt</listchoice>
    <listchoice command="cmdfontsize2">2 pt</listchoice>
    <listchoice command="cmdfontsize1">1 pt</listchoice>
  </selections>
</command>
```

Element Hierarchy

```
<config>
  <features>
    <standard>
      <font size>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, the font size is not used.
localeref	String	""	The identifier to translate the #text description.
name	String	"3"	The name of the font. It must be a value from 1 through 7. 1 is the smallest font, 7 is the largest. Fonts of any other names are not used.
#text	String	""	The text that defines the font. This text appears on the dropdown list but is not inserted into the font tag.

headings

The section that specifies the heading levels for paragraphs.

Element Hierarchy

```
<config>
  <features>
    <standard>
      <header level>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the heading listing is not used.

heading[x]

Defines a heading level. The [x] value in the name must be between, and including, 1 and 6. Any other values are not read in.

These are the heading names.

- heading1
- heading2
- heading3
- heading4
- heading5
- heading6

Remarks

Below are the heading commands available to list boxes and buttons. These commands control the block header type.

- cmdheading1
- cmdheading2
- cmdheading3
- cmdheading4
- cmdheading5
- cmdheading6
- cmdheadingStd (returns text to normal)

They do not need to be defined in the feature as commands. They are available for list box commands and scripting.

Although you can use the heading[x] element to define block header levels, the preferred method is to use a **selections** element to generate a list of options with commands.

Here is an example of the preferred way of defining available header levels.

```
<command name="cmdheaderlevel" style="listbox" visible="true">
  <image key="headerlevel" />
  <caption localeRef="btntxthdrlvl1"></caption>
  <toolTipText localeRef="btnhdrlvl1">Set the header level</toolTipText>
  <selections name="headinglist" enabled="true" sorted="true">
    <listchoice command="cmdheading1" localeref="hdgtxtlvl1">Heading 1</listchoice>
    <listchoice command="cmdheading2" localeref="hdgtxtlvl2">Heading 2</listchoice>
    <listchoice command="cmdheading3" localeref="hdgtxtlvl3">Heading 3</listchoice>
    <listchoice command="cmdheading4" localeref="hdgtxtlvl4">Heading 4</listchoice>
    <listchoice command="cmdheading5" localeref="hdgtxtlvl5">Heading 5</listchoice>
    <listchoice command="cmdheading6" localeref="hdgtxtlvl6">Heading 6</listchoice>
    <listchoice command="cmdheadingstd" localeref="hdgtxtnorm">Normal</listchoice>
  </selections>
</command>
```

Element Hierarchy

```
<config>
  <features>
    <standard>
      <headings>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the heading listing is not used.
Localeref	String	""	The translation of the #text description.
Name	String	""	These commands control the header levels. Heading1 - "heading 1" Heading2 - "heading 2" Heading3 - "heading 3" Heading4 - "heading 4" Heading5 - "heading 5" Heading6 - "heading 6"
#text	String	""	Text description of the header. This value is included in the header level listing.

External Features

Description

Use the external feature of the configuration data to extend the editor by defining external client functionality. This includes applications, JavaScript, and Visual Basic (VB) script.

You use the command element to define commands that execute the external code.

Element Hierarchy

```
<config>
  <features>
    <external>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, external commands are disabled.

Adding External Features

You can quickly add functionality to **eWebEditPro+XML** using JavaScript or Visual Basic. External commands defined are sent up as external events. This is a powerful way to define features without requiring the development of binary modules.

See Also: “[Custom Commands](#)” on page 215

Follow these guidelines when creating external features.

- Define the new functionality as a command within the `<external>` section of the configuration data.
- This section acts as the definition for the External Event feature.
- Follow all rules for defining standard features.

Examples

Here is an example of the external script/client command definition.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="4" revision="1">
  <features>
    <external enabled="true">
      <!--This is a user defined command that will go back up to the script-->
```

```
<command name="saveaspdf" style="0" visible="true">
    <image key="http://site.com/images/pdf.gif"/>
        <caption localeRef="btnScrPdf"> Safe as a PDF file.</caption>
        <toolTipText localeRef="btnPdf">Saves as PDF.</toolTipText>
    </command>
    .
    .
</external>
</features>
.
.
```

Here is an example of a custom module creating its own section within the features.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro" version="4" revision="1">
<features>
    <pdfgenerator enabled="true">
        <!--This is a user defined command sent to a DLL-->
        <command name="saveaspdf" style="0" visible="true">
            <image key="http://site.com/images/pdf.gif"/>
            <caption localeRef="btnScrPdf">Safe as a PDF file.</caption>
            <toolTipText localeRef="btnPdf">Converts to PDF.</toolTipText>
        </command>
    </pdfgenerator>
</features>
```

Form Elements

Description

Let the user create an HTML form.

See Also: In the **eWebEditPro+XML** User Guide: “Form Elements Toolbar” in the chapter “Toolbar Buttons”

Element Hierarchy

```
<config>
  <features>
    <formelements>
```

Attributes

Command	External Name	Description
cmdformform	Form	Inserts opening and closing form tags. For example: <pre><form name="Test" action="http://localhost/ewebeditpro5/formtest.htm" method="post"> </form></pre>
cmdformbutton	Button	Inserts a button. For example: <pre><input type="button" value="Test Button" name="Test" /></pre>
cmdformsubmit	Submit Button	Inserts a submit button. For example: <pre><input type="submit" value="Submit" /></pre>
cmdformreset	Reset Button	Inserts a reset button. For example: <pre><input type="reset" value="Reset Page" /></pre>
cmdformhidden	Hidden Text Field	Inserts a hidden text field. For example: <pre><input type="hidden" value="This is initial content" name="mycontent" /></pre>
cmdformtext	Text Field	Inserts a text field. For example: <pre><input size="15" value="This is initial content" name="mycontent" /></pre>
cmdformpassword	Password Field	Inserts a password field. For example: <pre><input type="password" value="" name="mypassword" /></pre>

Command	External Name	Description
cmdformtextarea	Textarea Field	<p>Inserts a textarea field. For example:</p> <pre><textarea name="mycontent" rows="5" cols="40">This is initial content</textarea></pre>
cmdformradio	Radio Button	<p>Inserts a radio button. For example:</p> <pre><input type="radio" checked="checked" name="mybutton" /></pre>
cmdformcheckbox	Checkbox	<p>Inserts a check box. For example:</p> <pre><input type="checkbox" checked="checked" name="mycheckbox" /></pre>
cmdformselect	Select	<p>Inserts a selection box. For example:</p> <pre><select multiple="multiple" size="25" name="myselectbox"> <option value="option1">option1</option> <option value="option2">option2</option> </select></pre>
cmdformfile	File Upload	<p>Inserts a File Upload field and a Browse button. For example:</p> <pre><input type="file" size="10" name="Save" /></pre>

Viewing and Editing HTML Content

This section describes elements that let users view and edit the HTML content of their Web page.

The ViewAs Feature

The ViewAs feature determines whether or not users can view the HTML source code. If you allow users to view source code, they do so by right-clicking the mouse while the cursor is in the editor. When they do, two menu options appear.

- **View HTML** - lets the user view the source code
- **View WYSIWYG** - returns the user to edit mode

If you allow users to view source code, you can further specify if they can view only the body of the page or the entire page including the header.

Disabling Custom Toolbar Buttons View as HTML Mode

The following JavaScript is an example of how to disable (or gray-out) custom toolbar buttons when the user selects "View As HTML". And, how to re-enable buttons when the user switches back to "View WYSIWYG".

1. Add the following to customevents.js or the page with the editor.

2. Specify the names of the commands in the myCustomCommands array.

```
var myCustomCommands = [ "jsmycommand1", "jsmycommand2",
"jsmycommand3" ];
function myUpdateButtonStatus(sEditorName, strCmdName, strTextData,
lData)
{
var bDisable = ("cmdviewashtml" == strCmdName);
var objInstance = eWebEditPro.instances[sEditorName];
var objMenu = objInstance.editor.Toolbars();
var objCommand = null;
for (var i = 0; i < myCustomCommands.length; i++)
{
objCommand = objMenu.CommandItem(myCustomCommands[i])
if (objCommand)
{
objCommand.setProperty("CmdGray", bDisable);
}
}
}
eWebEditProExecCommandHandlers[ "cmdviewashtml" ] =
myUpdateButtonStatus;
eWebEditProExecCommandHandlers[ "cmdviewaswysiwyg" ] =
myUpdateButtonStatus;
```

Element Hierarchy

```
<config>
  <features>
    <viewas>
```

Child Elements

cmd

Attributes

See Also: "[<Viewas> Attribute: View](#)" on page 602

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, users cannot view the content in different formats.
mode	String	Body	Specifies how much of the source code appears when the user views HTML. This attribute has two values. "Body" - only the body of the document "Whole" - the entire source, including headers
publish	String	Cleanhtml	The level of cleanliness applied when the user chooses ViewHTML. The higher the level, the potentially more time to process the source. This attribute has three values. <u>Important:</u> If you are using eWebEditPro+XML with an Ektron CMS, leave this setting as xhtml . "Minimal" - General tag organization "Cleanhtml" - Removes overlapping tags and merges font tags "Xhtml" - HTML level of organization
unicode	Boolean	False	If true, Unicode characters appear as their character reference (for example, ֪). Otherwise, they appear as question marks (?). See Also: " Viewing and Saving Unicode Characters " on page 418

The EditHTML Feature

The EditHTML feature determines whether or not users can edit the HTML source of the content by right clicking the mouse and choosing **Insert HTML**. If content is selected when the users clicks **Insert HTML**, that content appears in the dialog box and can be edited.

Users can also use **Insert HTML** to enter an HTML fragment at the current cursor location.

NOTE Even if you set `edithtml` to false, users can edit the HTML source via the ViewAs feature. See Also: "The ViewAs Feature" on page 390

Element Hierarchy

```
<config>
  <features>
    <edithtml>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	True	If false, the editHTML feature is disabled.

Cleaning HTML

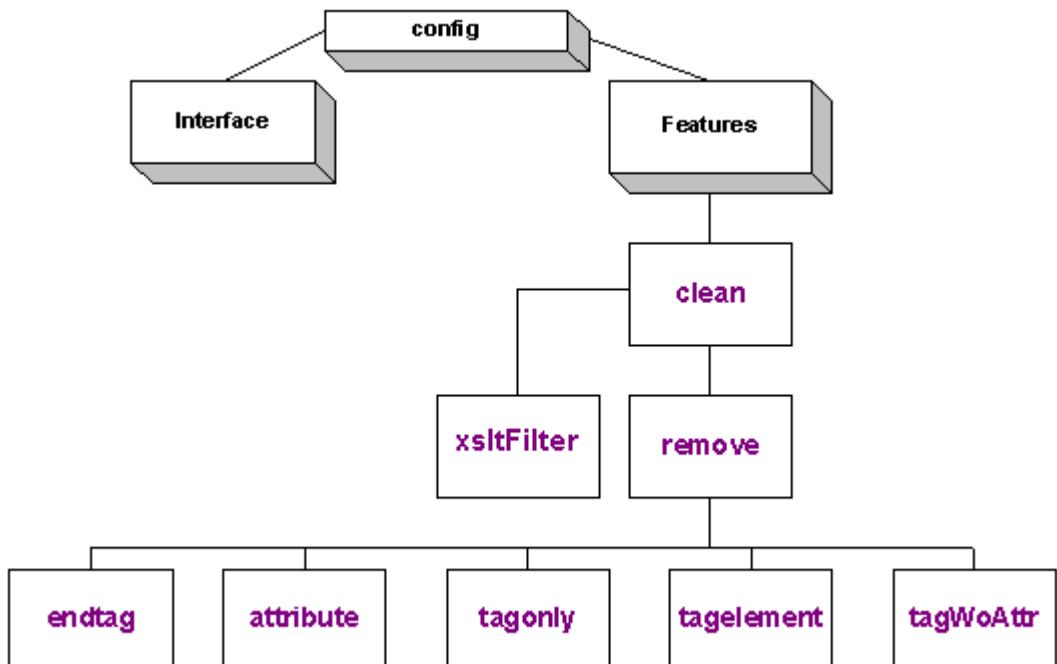
eWebEditPro+XML provides the following elements that prepare the HTML code of the Web content for publishing.

- clean
- remove, a sub element of clean
- endtag, attribute, tagonly, tagWoAttr and tagelement: sub-elements of remove

The clean feature defines general HTML clean-up features, such as the quality of the HTML code output by the editor.

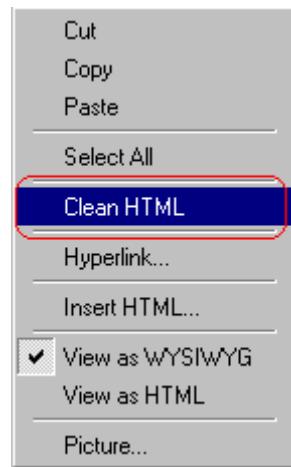
You can use Remove, Endtag, Attribute, Tagonly, Tagelement, and TagWoAttr elements to remove specific elements from the content when it is cleaned.

Clean Element Hierarchy



Providing User Access to the Clean Feature

By default, the command that launches the clean feature (`cmdclean`) appears on the context-sensitive menu.



However, you can assign the command to a button, just as you can assign any other command to a button (see “[Commands](#)” on page 195).

The user also receives an option to clean HTML when pasting content from Microsoft Word 2000.

Clean Element

The clean HTML feature ensures that the content loaded into the editor is readable and concise HTML source code.

Element Hierarchy

```
<config>
  <features>
    <clean>
```

Child Elements

```
remove
xsltFilter
```

Attributes

Name	Attribute Type	Default	Description
charencode	String	"charref"	<p>Determines how to encode special and extended characters. Five values are available.</p> <ul style="list-style-type: none"> • binary • entityname • charref • special • latin <p>For more information, see "Encoding Special Characters" on page 417. See Also: "preservechars" on page 398</p>
cr	String	"cr"	<p>How to translate the carriage return character when the content is saved. Four values are available.</p> <p>"" - remove cr character "cr" - do not process cr characters "charref" - replace cr character with its character reference, which is &#13 "\r" - replace cr character with \r</p>

Name	Attribute Type	Default	Description
feedbacklevel	String	0	<p>Lets you control when the cleaning dialog displays. The dialog notifies the user that eWebEditPro+XML is cleaning content and has not crashed.</p> <hr/> <p>Note: Depending on the speed of the client system and the size of the content, cleaning can last from several seconds to several minutes.</p> <hr/> <p>The attribute can have one of three values:</p> <p>0 (default value) - Normal display of clean dialog. The cleaning dialog displays if processing might take more than three seconds. This estimate considers the following factors: large content, XML tag processing, image file manipulation, MS Word content cleaning, and style sheet processing.</p> <p>1 - Only display clean dialog if document is large. If the raw content exceeds the value set in the <code>showonsize</code> attribute of the <code>clean</code> element, the cleaning dialog appears when cleaning is performed. The dialog does not appear under other processing, such as XML tag processing, image file manipulation, MS Word content cleaning, style sheet processing, or other functionality.</p> <p>See Also: "showonsize" on page 399</p> <p>2 - Never display clean dialog. During cleaning, the user interface and possibly the browser become unresponsive or sluggish, and the user is not notified why.</p>

Name	Attribute Type	Default	Description
hideobject	Boolean	"true"	<p>If the value is true, the object tag is hidden and protected from the DHTML control. (The object tag is the HTML tag that loads an object, such as an ActiveX control, in a browser.)</p> <p>If the value is false, the object tag is not hidden and the control tries to render the object to the user. If the object does not exist, the editor pauses for a long time until the operation times out. While the editor is attempting to render, the user cannot interact with it.</p> <p>The default value (true) hides the tag so there is no chance that the editor will get "stuck."</p> <p>Here is an example setting.</p> <pre><clean hideobject="true" charencode="charref" cr="cr" lf="lf" showonsize="5000" preferfonttag="false" reducetags="true" showdonemsg="true" prompt="true"> <remove> <tagWoAttr>SPAN</tagWoAttr> </remove> </clean></pre>
If	String	"If"	<p>How to translate the line feed character when the content is saved. Four values are available.</p> <p>"" - remove If character</p> <p>"If" - do not process a If character</p> <p>"charref" - replace If character with its character reference, that is &#10;</p> <p>"\n" - replace If character with \n</p>
preferfont	Boolean	"false"	<p>If true, span tags with font styles are converted to font tags.</p> <p>If a font name, color, or size are specified using a span tag (for example, in content pasted from MS Word), the span tag can be converted to a font tag. Font tags are compatible with older browsers and allow font attributes to be easily edited in eWebEditPro+XML.</p>
mswordfilter	Boolean	"false"	<p>If true, converts Word formatting to an HTML format where possible. For example, Word's Heading 1 style is converted to a set of <h1> tags.</p>

Name	Attribute Type	Default	Description
preservechars	String	""	<p>Identifies characters that will not be converted to character references. Conversion of these characters is done on the server side only.</p> <p>When entering values for this element, enter character references if XML requires them. Refer to an XML reference to determine which characters require conversion, and how to convert them.</p> <p>For example, to prevent the “less than” (<) and “greater than” (>) characters from being converted to their character references, enter</p> <pre>preservechars value="&lt;&gt;"</pre> <p>For more information about special characters, see “Encoding Special Characters” on page 417.</p>
prompt	Boolean	“True”	<p>Can suppress the message that appears when pasting content from Microsoft Office 2000 or later: HTML code generated by Office 2000 has been detected...Do you want to clean the HTML code now?</p> <p>To suppress the message, set the value to <code>false</code>.</p> <p>The <code>autoclean</code> attribute of the <code><standard></code> element determines whether eWebEditPro+XML attempts to clean content pasted from Word.</p> <p>See Also: “autoclean” on page 355;</p>
reducetags	Boolean	“false”	<p>Whether eWebEditPro+XML eliminates unnecessary tags.</p> <p>When a user pastes content from other applications into eWebEditPro+XML, the content may contain redundant tags, such as extra font and bold tags. If this is set to <code>true</code>, extra tags are combined or safely removed.</p>

Name	Attribute Type	Default	Description
showonsize	integer		<p>The minimum number of characters of HTML code needed to display a dialog box that appears when the user saves content. The dialog indicates that eWebEditPro+XML is cleaning HTML.</p> <p>This attribute prevents the dialog box from displaying when there is little or no content.</p> <p>See Also: "feedbacklevel" on page 396</p> <hr/> <p>Note: This attribute does not appear in the configuration data by default. You must enter the attribute name and value to use it.</p> <hr/>
showdonemsg	Boolean	"false"	<p>Suppresses the message dialog box that appears after cleaning: "The cleaning of the HTML source is complete".</p> <p>You can also suppress the message when invoking the clean command via JavaScript, but have it appear when a user cleans content using the context menu. To suppress the message only when calling the clean command in JavaScript, pass a numeric data argument of 1. In JavaScript:</p> <pre>eWebEditPro.instances[0].editor.ExecCommand("cmdClean", "", 1);</pre>

Remove Element

This rule defines what elements are removed from the content when it is cleaned.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
```

Child Elements

`endtag`, `attribute`, `tagonly`, `tagelement`, `tagWoAttr`

Attributes

Name	Attribute Type	Description
		There are no attributes to the remove element.

Endtag Element

This attribute defines which elements are globally removed from content when it is cleaned. In general, this option is not recommended. But, there may be situations in which certain end tags (for example, </p>) are not desired and can be removed with little risk.

eWebEditPro+XML removes end tags when the content is saved.

This option is ignored if the `publish` attribute of the `standard` element is set to **xhtml**.

See Also: "publish" on page 355

You can only enter one `<remove>` element, but you can enter several `<endtag>` elements. You must enter one set of `<endtag>` elements for every tag to be removed.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
        <endtag>
```

Attribute

Name	Attribute Type	Description
#text	String	The tag to remove.

Example

```
<clean cr="cr" lf="lf" autodetect="yes">
  <remove>
    <endtag>p</endtag>
    <endtag>li</endtag>
  </remove>
</clean>
```

Attribute Element

When the user cleans the content, that procedure can remove specified attributes from the content. In general, this option is not recommended, but there may be situations in which you want to remove certain attributes (for example, id, onclick, etc.).

eWebEditPro+XML removes attributes when the content is saved.

This option is ignored if the `publish` attribute of the standard `element` is set to `xhtml`.

See Also: "publish" on page 355

You can only enter one `<remove>` element, but you can enter several `<attribute>` elements within it. You must enter one set of `<attribute>` elements for every tag to be removed.

Element Hierarchy

```
<config>
  <features>
    <clean>
      <remove>
        <attribute>
```

Attribute

Name	Attribute Type	Description
#text	String	The attribute to remove.

Example

```
<clean cr="cr" lf="lf" autodetect="yes" >
  <remove>
    <attribute>onclick</attribute>
  </remove>
</clean>
```

Tagonly and Tagelement Elements

When the user cleans the content, that procedure can remove specified HTML tags only, or specified tags along with any content between them.

For example, you can set up the clean element to remove all font tags, image (img) tags, and script elements.

You can only enter one `<remove>` element, but you can enter several `<tagonly>` and `<tagelement>` elements within it.

Element Hierarchy

```
<config>
```

```

<features>
  <clean>
    <remove>
      <tagonly>
      <tagelement>

```

Attribute

Name	Attribute Type	Description
#text	String	The tag to remove.

Example

```

<clean cr="cr" lf="lf" autodetect="yes" >
  <remove>
    <tagonly>font</tagonly>
    <tagelement>script</tagelement>
  </remove>
</clean>

```

TagWoAttr Element

When the user cleans the content, the cleaning can remove specified tags that have no attributes. Use the `<tagWoAttr>` element to accomplish this.

For example, you can use `<tagWoAttr>` to remove all SPAN tags with no attributes. If the cleaning finds SPAN tags *with* attributes, those tags are not affected.

You can enter only one `<remove>` element, but you can enter several `<tagWoAttr>` elements within it.

Element Hierarchy

```

<config>
  <features>
    <clean>
      <remove>
        <tagWoAttr>

```

Attribute

Name	Attribute Type	Description
#text	String	The tag to remove.

Example

```

<clean cr="cr" lf="lf" autodetect="yes" >
  <remove>

```

```
<tagWoAttr>SPAN</tagWoAttr>
</remove>
</clean>
```

xsItFilter Element

This element is to be used with the Data Designer.

The `xsItFilter` element identifies an XSLT file that can modify content by removing or replacing specific tags and attributes. Here is the default value:

```
<xsltFilter src="[eWebEditProPath]ektfilter.xslt"/>
```

NOTE

[eWebEditProPath] refers to the eWebEditProPath variable in the `ewebeditpro.js` file.

You can implement almost any custom change to data design content by modifying the `xsItFilter` file. The following commonly-requested changes are built into `ektfILTER.xslt`, although they are commented out by default. To enable the changes, remove the comment markers.

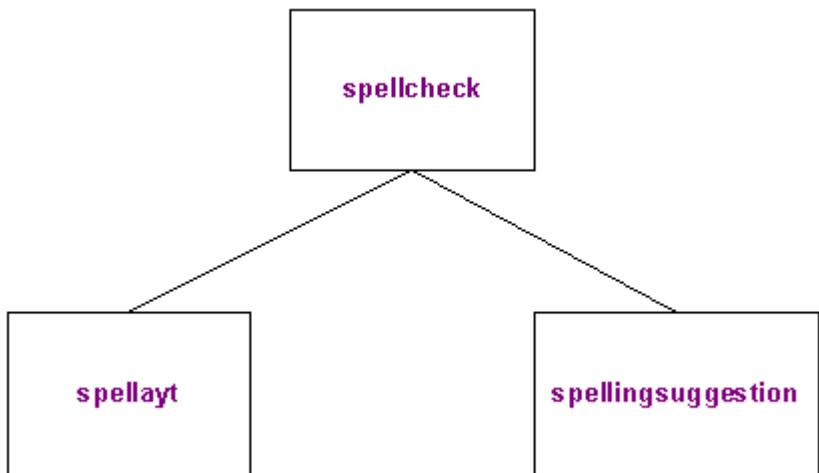
The `ektfILTER.xslt` file fixes the `align=center` to `align=middle` problem. It also has `include` statements that refer to these XSLT files.

- `ektfILTERekttags.xslt` - processes the following custom XML tags:
 - `ekt_date` (displays current date)
 - `ekt_toc` (creates table of contents using h1-h6 tags; the table of contents entries can be either numbers or an outline)
- `ektfILTERxhtml10.xslt` - ensures content complies with XHTML 1.0 transitional by removing browser-specific tags and attributes
- `ektfILTERcustom.xslt` - commonly requested custom filtering, which makes the following replacements
 - remove `<DIV>` tags within `` tags
 - convert `` to ``
 - convert `<I>` to ``

The Spellcheck Feature

The spellcheck feature controls the operation of spell checking within the client. Note that user can perform spell checking on demand or “as you type.” For more information about the user interaction with spell checking, see the “Checking Spelling” section of the **eWebEditPro+XML** User Guide.

The feature has three elements, which are depicted on the following chart and then summarized in a table.



Command	Lets You Specify
Spellcheck	<ul style="list-style-type: none">Activation of the spell check featureThe language of spelling dictionaryWhether to use spell checker without MS WordA primary and secondary spell checker
Spellayt	<ul style="list-style-type: none">What triggers spell check as-you-typeImage file that marks misspelled wordsDelay between cycles
Spellingsuggestion	<ul style="list-style-type: none">Number of correctly spelled words similar to misspelled word that appear

See Also: [“Setting the Language of Spell Checking” on page 287](#)

Spellcheck

Defines whether or not the spell check feature operates, the language, and an alternative to using MS Word for the dictionary.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
```

Child Elements

spellayt, spellingsuggestion

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false, spell checking is disabled.
langid	String	0	A Microsoft Word Locale ID (LCID) that identifies a particular language. For example, the LCID for English is 1033, and the LCID for Japanese is 1041. Use this attribute to change the spelling dictionary that eWebEditPro+XML refers to. If you leave the default value (0), the spell check refers to the language selected in Microsoft Word. Before a client PC can refer to a foreign dictionary, that language must have been installed on the PC. For more information, see the following article on Microsoft's Web site: "WD2000: Supported Language ID Reference Numbers (LCID)" http://support.microsoft.com/default.aspx?scid=kb;en-us;221435

Name	Values	Default	Description
dictionary or dictionary2	String	dictionary2= "WinterTreeSC2.CWinterTreeSC"	<p>This attribute provides a spell check capability that does not require Microsoft Word on a client. As a result, users can spell check content without MS Word. Also, if only some users have MS Word, you can identify a primary and secondary spell checker. For example, you can set MS Word as the primary spell checker and the alternate spell check software as the secondary.</p> <p>If you identify a primary and secondary spell checker, eWebEditPro+XML first attempts to use the primary. If eWebEditPro+XML cannot find the primary, it uses the secondary spell checker.</p> <p>Both spell checkers refer to any custom dictionaries created in MS Word. However, the alternate spell checker only refers to an English dictionary -- it cannot spell check foreign text as MS Word can. (See Also: "langid" on page 405)</p> <p>Controlling this Feature</p> <p>Administrators control this feature through the dictionary and dictionary2 attributes of the spellcheck element. If you want MS Word to be used by default and only use the alternate when MS Word is unavailable, use this syntax:</p> <pre><spell check dictionary2=EkWinterTreeSC2.CWinterTreeSC"></pre> <p>If you want the alternate spell checker to be used by default and only use MS Word when the alternate is unavailable, use this syntax:</p> <pre><spell check dictionary= "WinterTreeSC.CWinterTreeSC"></pre> <p>The Client Installation File</p> <p>You must install the SpellChecker to every client system that will use this feature. This lightweight client installation file associates the alternate spell checker and its dictionary.</p> <p>To download the client installation file, go to http://www.ektron.com/support/downloads/ewebeditpro/wintertree/spellcheckercomp.exe.</p>

Spellayt

Defines how spell checking as-you-type operates.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
      <spellayt>
```

Attributes

Name	Values	Default	Description
autostart	Boolean	True	If true , spell check starts to check spelling “as-you-type” as soon as possible without user intervention. The editor is slower to launch due to spell checking. If false , the user must press the button or toolbar menu option to activate spell check as-you-type.
enabled	Boolean	True	If false , auto-spell checking is disabled.
markmisspelledsrc	String	""	Specifies the URL of the graphic file (by default, a wavy red line) that marks misspelled words. The name of the file provided is wavyred.gif. The default value ("") resolves to the location of the configuration data. This is interpreted as a path, so the case is maintained.
delay	String	20	Auto spellcheck continually checks all of the words in the editor’s content, from top to bottom. This attribute sets the number of milliseconds that the auto spellcheck feature waits when it reaches the end of the content before restarting. If you set a low value (such as the default, 20), the spellcheck’s performance improves but more CPU resources are required. If you set a high value, the spellcheck’s performance degrades but more CPU resources are available.

Example

```
<spellayt autostart="false" markmisspelledsrc="[eWebEditProPath]/wavyred.gif" delay="20" />
```

Spelling suggestion

Defines suggestions for correcting spelling errors when user is using spell checking “as-you-type.”

NOTE These settings only take effect when spell checking on demand is being used.

Element Hierarchy

```
<config>
  <features>
    <spellcheck>
      <spellingsuggestion>
```

Attributes

Name	Values	Default	Description
enabled	Boolean	True	If false , the spell checker does not suggest replacement words when user is using spell checking “as-you-type.”
max	Integer	20	The maximum number of correctly-spelled words that appear after the spell checker finds a misspelled word when user is using spell checking “as-you-type.” To view this list, right click the mouse.

Example of Spell Check Features

The following is the default version of the spell check features in the configuration data.

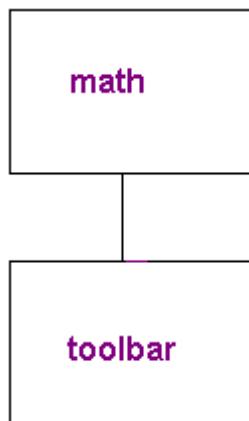
```
<spellcheck langid="0" enabled="true" dictionary2="WinterTreeSC.CWinterTreeSC">
  <spellayt autostart="false" markmisspelledsrc="[eWebEditProPath]/wavyred.gif" delay="20"/>
  <spellingsuggestion enabled="false" max="4"/>
  <cmd name="cmdspellayt" key="spellayt" ref="cmdSplayt" style="toggle"/>
  <cmd name="cmdspellcheck" key="spellcheck" ref="cmdSplck"/>
</spellcheck>
```

The Math Expression Editor Feature

The math feature controls the operation of the Math Expression editor.

For more information about the user interaction with the Math Expression editor, see the “Working with the Math Expression Editor” section of the **eWebEditPro+XML** User Guide.

The feature has one element, depicted on the following chart.



Math

Defines the Math Expression editor. You can determine the image file format in which expressions are saved.

Element Hierarchy

```
<config>
  <features>
    <math>
```

Child Elements

toolbar

Attributes

Name	Values	Default	Description
imagetype	string	png	Determines the file format in which the expression is saved. Choices are png and jpg.

Toolbar

Use this element to insert the WebEQ toolbar configuration. If none is given, the standard toolbar is displayed.

Details of WebEQ toolbar customization are provided by Design Science, Inc. For more information, go to <http://www.dessci.com/en/products/webeq/interactive/inputctrl.htm>.

Element Hierarchy

```
<config>
  <features>
    <math>
      <toolbar>
```

Attributes

none

Example of Math Features

The following is the default version of the spell check features in the configuration data.

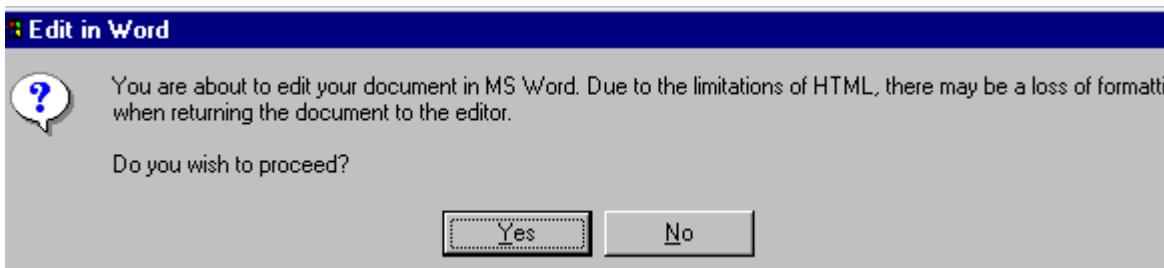
```
<math imagetype="png">
  <!-- imagetype may be "png" or "jpg" -->
  <cmd name="cmdmath" key="math" ref="cmdMath" />
  <toolbar></toolbar>
</math>
```

Editing in Microsoft Word

The `msword` element enables the Edit in Word toolbar button ()¹, which lets users perform all editing functions within Microsoft Word®. The client computer must have Microsoft Word 2000 or greater.

Users may prefer to edit within Word because of familiarity with Word's user interface, and to use additional functionality available in Word.

This element launches the `cmdmsword` command, which checks the value of the `warn` attribute. If the attribute's value is `true`, the following warning displays.



If the user elects to proceed, Microsoft Word launches. Any content in **eWebEditPro+XML** is copied to a temporary Word document. The user then edits within Word.

When done, the user either closes Word (using the small **x** in the top right corner of the window) or returns to **eWebEditPro+XML** and clicks the Word button () again. The Word content is copied back into **eWebEditPro+XML**.

When Word content is pasted into **eWebEditPro+XML**, the Clean HTML Code dialog box appears, asking the user if he wants to clean excessive HTML code.

See Also: "[Cleaning HTML](#)" on page 393

Element Hierarchy

```
<config>
  <features>
    <msword>
```

Child Elements

```
  cmd
```

Attributes

Name	Attribute Type	Default	Description
warn	Boolean	True	Determines whether a warning displays when user launches Word and returns to eWebEditPro+XML from Word.
startupmode	string	htmlview	<p>Determines Word's initial view format. Choose from these options:</p> <ul style="list-style-type: none"> • normalview - a document formatted on a simplified page • htmlview - a document as it appears in a Web browser • wordview - a document as it appears when you print it • readingview - a document in full page view • outlineview - a document that shows hidden characters as well as visible ones <p>After the document loads, the user can change the view using the menu options.</p>

Using the Long Parameter with cmdmsword

If you send the msword command via JavaScript, you can use the long parameter to specify whether you want MS Word started or shut down.

- If you specify 1 in the long parameter, MS Word launches
- If you specify 0, MS Word shuts down

Here is an example of it starting MS Word.

```
<input type="button" value="Run Word"
onClick="eWebEditPro.instances['MyContent1'].editor.ExecCommand('cmdmsword', '', 1)">
```

Here is an example of it shutting down MS Word.

```
<input type="button" value="Run Word"
onClick="eWebEditPro.instances['MyContent1'].editor.ExecCommand('cmdmsword', '', 0)">
```

How Microsoft Word Content is Processed

There are three ways to handle Microsoft Word content.

- **Conserve formatting** from Microsoft Word wherever possible.

- Convert Word styles to standard HTML where possible.
- Conform to style sheet by discarding some Word styles.

Each method is explained, with their pros and cons and how to configure them.

Conserve Word Formatting

This approach preserves Word formatting where possible. It is impossible to retain all formatting because the HTML standards do not support all of Word's formatting features. Also, Word uses CSS styles that are not available to the **eWebEditPro+XML** when copying and pasting from the clipboard.

You will retain more or Word's formatting if you specify a style sheet file (.css) that duplicates the styles used in Word. To make this task easier, **eWebEditPro+XML** provides the ektnormal.css file, which is based on MS Word 2000's Normal.dot style template.

Pros

- Preserves as much Word format as possible

Cons

- Word styles may cause problems when the user tries to change formatting in **eWebEditPro+XML**. For example, if an inline style attribute is used to underline text, clicking **eWebEditPro+XML**'s underline button has no effect.
- Content is large due to inline style attributes
- Will probably display differently in older browsers

Configure

In your configuration XML data (for example, config.xml), set the following attributes.

```
<clean preferfonttag="false" reducetags="true" mswordfilter="false">
  <remove>
    <tagWoAttr>SPAN</tagWoAttr>
  </remove>
</clean>
<standard autoclean="false" ...>
  <style publishstyles="false" href="[eWebEditProPath]/ektnormal.css"
  preservewordstyles="true" preservewordclasses="true">
  </style>
```

Ensure that `mswordfilter` and `autoclean` are **false**. Both attributes may remove Word formatting. Also, ensure both `preserveword` attributes are **true**; otherwise, Word formatting is lost.

Options

If you want to retain even more HTML tags from Word, set `reducetags="false"`.

If you want to display the content without using a style sheet, like ektnormal.css, set `publishstyles="true"`. Once you do this, it can be difficult to change the format later.

Convert Styles

This approach tries to preserve Word formats, but converts the content to standard HTML that is easier to edit in **eWebEditPro+XML**. Formatting may be lost to meet this goal. This approach is recommended when converting Word documents to Web content.

Pros

- Suitable for editing in **eWebEditPro+XML**
- Reduces content size
- More standards compliant

Cons

- Formatting may be lost

Configure

In your configuration XML data (for example, config.xml), set the following attributes.

```
<clean preferfonttag="false" reducetags="true" mswordfilter= "true">
    <remove>
        <tagWoAttr>SPAN</tagWoAttr>
    </remove>
</clean>
<standard autoclean="true" ...>
    <style publishstyles="false" href="[eWebEditProPath]/ektnormal.css"
    preservewordstyles="false" preservewordclasses="true">
    </style>
```

Ensure that `mswordfilter` is **true**. This attribute converts Word formatting. Also, ensure that `preservewordstyles` is **false**. Otherwise, it may be difficult to edit in **eWebEditPro+XML**. In general, it is safe to set `preservewordclasses` to **true** because the ektnormal.css style sheet retains most Word styles without sacrificing ease of use.

Options

If you want to use FONT tags where applicable, set `preferfonttag="true"`. If you want to always remove style attributes, even if the content does not come from MS Word, add `style` to the `<remove>` element, as shown below.

WARNING!

This also removes the background color.

```
<remove>
    <attribute>style</attribute>
```

```
<tagWoAttr>SPAN</tagWoAttr>
</remove>
```

Conform by Discarding

This approach controls formatting through external style sheets (.css files). Your Web site's styles supersede formatting applied by a user in Word. Simple formatting like bold and italic are usually allowed, but Word-specific styles and style classes are removed.

Pros

Conforms to preferred styles where possible

Cons

Loss of some formatting

Configure

In your configuration XML data (for example, config.xml), set the following attributes.

```
<clean preferfonttag="false" reducetags="true" mswordfilter= "true">
  <remove>
    <tagWoAttr>SPAN</tagWoAttr>
  </remove>
</clean>
<standard autoclean="true" ...>
  <style publishstyles="false" href="mystylesheet.css" preservewordstyles="false"
preservewordclasses="false">
  </style>
```

Ensure that `mswordfilter` is **true**. This attribute that converts Word formatting. Also, ensure that `preservewordstyles` and `preservewordclasses` are **false**. Setting `preservewordclasses` to false removes all the `class="Mso..."` attributes. Typically, you replace ektnormal.css with your own style sheet.

Options

If you want to always remove FONT and U tags, add them to the `<remove>` element, as shown below.

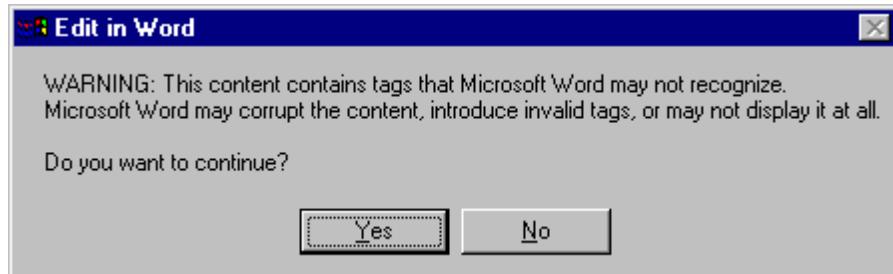
```
<remove>
  <tagonly>FONT</tagonly>
  <tagonly>U</tagonly>
  <tagWoAttr>SPAN</tagWoAttr>
</remove>
```

Using Word to Edit XML Documents

Microsoft Word does not support the editing of XML documents. If a full XML document is loaded, the Word button () is disabled. Therefore, you should

disable the `msword` element for users who create and edit XML using **eWebEditPro+XML**.

If a Word document includes some custom/XML tags, the following dialog appears, warning the user of the problem.



The user can proceed and edit using Word or decide not to edit the document using Word.

Encoding Special Characters

Factors that Affect the Display of Special Characters

The HTML specification defines special characters for a set of punctuation symbols, accented letters, and a variety of non-Latin characters. As the HTML specification has changed, so has browser support for special characters.

For example, Microsoft defined several special characters that previously displayed only in Internet Explorer on Windows. They are extended characters that map to binary values 128 to 159. Depending on the browser version and operating system, the characters may appear as expected, as a question mark (?), or as a small rectangle. The W3C adopted most extended characters in HTML 4, but mapped them to different binary values.

Using the wrong font can also prevent the proper display of a character. This is a common problem when copying from Microsoft Word, where many special characters are in the Symbol font. If the font is not available in the browser or not permitted in the editor, special characters do not display properly.

For example, the Euro symbol was designed for the European Economic Community (EEC) in the late 1990s. Obviously, operating systems and browsers created earlier could not display it.

Euro character (shown using an image)	€
Euro in Verdana font (display depends on your browser)	€
Euro in Courier New font (display depends on your browser)	€
Entity Name	€
Microsoft Windows Extended Character Reference	€
HTML 4 Character Reference	€

Characters with binary values 160 to 255 are also special characters because they display differently depending on the browser's language (or locale) and the charset attribute in the meta tag on the Web page. Below is an example meta tag.

```
<meta http-equiv=Content-Type content="text/html; charset=iso-8859-2">
```

The display of special characters can also be controlled from the browser. For example

- in IE 5, from the menu bar, select **View > Encoding** > language of your choice. (You may need to install the IE option for international language support).
- in Netscape 4.7, select **View > Character Set** > language of your choice

In each case, the possible languages are grouped as West European (Latin1), East European (Latin2), Cyrillic, Arabic, Greek, Hebrew, and more. Each character set is defined by ISO 8859, a standard for coded graphic character sets established by the International Organization for Standardization.

The ISO 8859 special characters are listed below. When viewed in a browser, these characters display differently if you change your browser's encoding.

í ö £ þ ¥ ¡ § “ © ª « ¬ - ® –
 ° ± ² ³ ´ μ ¶ . , ¹ º » ¼ ½ ¾ ¸
 À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
 Đ Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý þ ß
 à á â ã ä å æ ç è é ê ë ì í î ï
 ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

In summary, the following factors affect the display of special characters.

- browser and browser version
- operating system
- language of the operating system (English, Polish, Arabic, etc.)
- font (Times, Arial, Helvetica, Symbol, etc.)
- charset attribute in the meta tag (windows-1252, iso-8859-1, etc.)
- encoding/character set setting of the browser (Western, Central European, UTF-8, etc.)

Viewing and Saving Unicode Characters

When a user views Web content in View As HTML mode, Unicode characters appear as their character reference (for example, ֪).

IMPORTANT!

The view source window can only display characters that match the system language of the operating system. That is, to display Japanese characters in source, the operating system must be Japanese Windows or Windows whose default system language is set to Japanese. Characters not supported by the operating system are converted to "?".

However, they do not need to be saved in that format. The charencode attribute in the clean element of the configuration data determines how Unicode characters are saved.

See Also: "charencode Attribute" on page 419

To save Unicode characters in a format other than character reference, set charencode to one of the following:

- **utf-8**, if using UTF-8 charset encoding
- **binary**, if using some other encoding, such as big5, shift_jis, etc.

Displaying Asian Languages

Many Asian languages, such as Japanese, Korean, and Chinese, are represented by two bytes instead of one. The binary values for these characters are in the range 256 to 65535. These are mapped as Unicode characters.

eWebEditPro+XML can optionally convert these characters to their character reference or leave them as double-byte binary Unicode values (not UTF-8). For example, a character whose binary value is 1234 converts to Ӓ.

Unicode Characters

Unicode characters (double byte characters typically used for Asian languages) are normally converted to character references, for example, Ӓ. To output Unicode characters as their double-byte binary value, set the charencode attribute to **binary**. If your site uses UTF-8 encoding, you can set the charencode attribute to **UTF-8** instead of binary, but the two are essentially the same.

Configuring for Extended and Special Characters

eWebEditPro+XML can be configured to represent extended and special characters in several ways. They are

- **binary** - extended, special, and double-byte characters as binary (Unicode, which can be converted to UTF-8)
- **entityname** - extended and special characters as their entity name; double-byte characters as their character reference
- **charref** - extended, special, and double-byte characters as their character reference. This is the default value.
- **special** - extended characters as their entity name; special characters as binary; double-byte characters as their character reference
- **latin** - extended characters as HTML 4 character references; special characters as binary; double-byte characters as their character reference

charencode Attribute

To configure **eWebEditPro+XML**, set the charencode attribute of the clean tag in the configuration data. For example,

```
<!-- values for charencode: utf-8, binary, entityname, charref, special, latin -->
<clean charencode="charref" ...>
```

NOTE

To prevent selected characters from being converted to character references, use the **preservechars** attribute of the clean element. For more information, see “**preservechars**” on page 398.

The values for charencode and their effect are described in the following table.

Value of charencode	Description	Sample
utf-8 or binary	<p>The sample shows all the characters with binary values 128 to 255. Characters 128-159 are extended characters. They are listed in two rows that start with 80 (the hexadecimal representation of 128) and 90. Characters 160-255 are special characters. They are listed in several rows that start with A0 (the hexadecimal representation of 160) through F0. The sample was captured using IE 5.0 on English language Windows (Latin1). Double-byte characters are not shown, but would be their binary value.</p> <p>WARNING: These characters only display properly if the operating system supports them. Even if they display in WYSIWYG mode, they may not display in View As HTML mode. If stored in a database, the database must support double-byte Unicode characters. May not be supported in Netscape Navigator 4.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <p>80: € ☁ , f „ … † ‡ ^ % œ Š \times œ Ž ☁</p> <p>90: ☁ ‘ ’ “ ” • — ~ ™ š > œ ☁ ž Ÿ</p> <p>Special Characters: (Latin1 shown)</p> <p>A0: ¡ ¢ £ ☁ ¥ § ª © º « ¬ ® ¯</p> <p>B0: ° ± ² ³ ´ μ ¶ ¸ ¹ º » ¼ ½ ¾ ¸</p> <p>C0: À Á Â Ã Ä Å Ç È É Ê Ë Ì Í Î Ï</p> <p>D0: Đ Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß</p> <p>E0: à á â ã ä å æ ç è é ê ë ì í î</p> <p>F0: ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ</p>

Value of charencode	Description	Sample
entityname	<p>Extended characters are represented using their entity name (for example, &euro;) where possible.</p> <p>Special characters are represented using their entity name (for example, &nbs; or Agrave;).</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre>80: &euro; &sbquo; &fnof; &bdquo; &hellip; &dagger; &Dagger; &circ; &permil; &Scaron; &lsaquo; &OElig; &#381; 90: &lsquo; &rsquo; &ldquo; &rdquo; &bull; &ndash; &mdash; &tilde; &trade; &scaron; &rsaquo; &oelig; &#382; &Yuml;</pre> <p>Special Characters: (Latin1 shown)</p> <pre>A0: &nbs; &iexcl; &cent; &pound; &curren; &yen; &brvbar; &sect; &uml; &copy; &ordf; &laquo; &not; &shy; &reg; &macr; B0: &deg; &plusmn; &sup2; &sup3; &acute; &micro; &para; &middot; &cedil; &sup1; &ordm; &raquo; &frac14; &frac12; &frac34; &iquest; C0: &Agrave; &Acute; &Acirc; &Atilde; &Auml; &Aring; &AElig; &Ccedil; &Egrave; &Eacute; &Ecirc; &Euml; &Igrave; &Iacute; &Icirc; &Iuml;</pre>
charref	<p>Extended characters are represented using their HTML 4 character reference (for example, &#8364;).</p> <p>Special characters are represented using their character reference (for example, &#160; or &#192;).</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre>80: &#8364; &#8218; &#402; &#8222; &#8230; &#8224; &#8225; &#710; &#8240; &#352; &#8249; &#338; &#381; 90: &#8216; &#8217; &#8220; &#8221; &#8226; &#8211; &#8212; &#732; &#8482; &#353; &#8250; &#339; &#382; &#376;</pre> <p>Special Characters: (Latin1 shown)</p> <pre>A0: &#160; &#161; &#162; &#163; &#164; &#165; &#166; &#167; &#168; &#169; &#170; &#171; &#172; &#173; &#174; &#175; B0: &#176; &#177; &#178; &#179; &#180; &#181; &#182; &#183; &#184; &#185; &#186; &#187; &#188; &#189; &#190; &#191; C0: &#192; &#193; &#194; &#195; &#196; &#197; &#198; &#199; &#200; &#201; &#202; &#203; &#204; &#205; &#206; &#207; D0: &#208; &#209; &#210; &#211; &#212; &#213; &#214; &#215; &#216; &#217; &#218; &#219; &#220; &#221; &#222; &#223;</pre>

Value of charencode	Description	Sample
special	<p>Extended characters are represented using their entity name (for example, &euro;) where possible.</p> <p>Special characters remain as binary, except the non-breaking space, which is represented as &nbsp;.</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre>80: &euro; &sbquo; &fnof; &bdquo; &hellip; &dagger; &Dagger; &circ; &permil; &Scaron; &lsaquo; &OElig; &#381; 90: &lsquo; &rsquo; &ldquo; &rdquo; &bull; &ndash; &mdash; &tilde; &strade; &scaron; &rsaquo; &oelig; &#382; &Yuml;</pre> <p>Special Characters: (Latin1 shown)</p> <pre>A0: &nbsp; ; &copy; &reg; &copy; &reg; &copy; &reg; &copy; &reg; B0: &deg; &pm; &plusmn; &mu; &Pound; &ordm; &ordm; &ordm; &ordm; C0: &Agrave; &Aacute; &Atilde; &Acirc; &Egrave; &Eacute; &Ecirc; &Igrave; &Iacute; &Icirc; D0: &Ograve; &Oacute; &Otilde; &Ocirc; &Otimes; &Ugrave; &Uacute; &Ucirc; &Ucirc; E0: &agrave; &aacute; &atilde; &acirc; &egrave; &eacute; &ecirc; &igrave; &iacute; &icirc; F0: &nbsp; &nbsp;</pre>
latin	<p>Extended characters are represented using their HTML 4 character reference (for example, &#8364;).</p> <p>Special characters remain as binary, except the non-breaking space, which is represented as &#160;.</p> <p>Double-byte characters are not shown, but would be their binary value.</p>	<p>Extended Characters: windows-1252 (WinLatin1)</p> <pre>80: &#8364; &#8218; &#402; &#8222; &#8230; &#8224; &#8225; &#710; &#8240; &#352; &#8249; &#338; &#381; 90: &#8216; &#8217; &#8220; &#8221; &#8226; &#8211; &#8212; &#732; &#8482; &#353; &#8250; &#339; &#382; &#376;</pre> <p>Special Characters: (Latin1 shown)</p> <pre>A0: &#160; ; &copy; &reg; &copy; &reg; &copy; &reg; &copy; &reg; B0: &deg; &pm; &plusmn; &mu; &Pound; &ordm; &ordm; &ordm; &ordm; C0: &Agrave; &Aacute; &Atilde; &Acirc; &Egrave; &Eacute; &Ecirc; &Igrave; &Iacute; &Icirc; D0: &Ograve; &Oacute; &Otilde; &Ocirc; &Otimes; &Ugrave; &Uacute; &Ucirc; &Ucirc; E0: &agrave; &aacute; &atilde; &acirc; &egrave; &eacute; &ecirc; &igrave; &iacute; &icirc; F0: &nbsp; &nbsp;</pre>

Choosing a charencode Value

The best charencode value to use depends on the environment in which the content is viewed and personal preference for entity names versus character references. Here are some examples.

- If the environment only supports 7-bit ASCII characters (for example, a database), you must use entityname or charref.

- Values of special or latin create smaller file sizes, because the special characters require one byte instead of six or more bytes to represent each character.
- A value of binary create the smallest file sizes for content that consists mostly of Asian characters (for example, Japanese, Korean, Chinese), because the characters require just two bytes instead of seven or more.
- Some sites convert Unicode characters to a byte stream format of UTF-8. If your site consistently uses UTF-8, use a value of utf-8.

The following table recommends `charencode` values for certain conditions.

Condition	Recommended charencode Value	Comments
Database supports only 7 bit characters	entityname or charref	Extended and special characters will be corrupted if wrong <code>charencode</code> value is selected. Your choice depends on your preference for entity names or character references.
Database supports only 8 bit characters	any except binary; use utf-8 only if your site uses UTF-8 consistently	Some special and all double-byte characters are corrupted if you choose binary.
Double-byte encoding, typically for an Asian language, and document size is important	binary or utf-8	Database must support Unicode (double-byte) characters. Note: Unicode is not the same as UTF-8.
Entity names are preferred	entityname	Extended and special characters are their entity name.
Entity names are preferred, but in a non-Western European language	special	Special characters are binary for different document encodings, but extended characters are their entity name.
ISO-8859 (Latin) or windows charset encoding on document, but not Latin1 (that is, not windows-1252 or iso-8859-1)	latin or special	Your choice depends on your preference for entity names or character references for extended characters.

Condition	Recommended charencode Value	Comments
Netscape Navigator 4 used for browsing	charref	Most extended and special characters appear. Double-byte characters do not appear if the browser or operating system does not support the language. If another charencode value is selected, some extended and special characters may appear as a question mark (?) or their entity name.
UTF-8 charset encoding on document	entityname or charref; use utf-8 only if your site uses UTF-8 consistently	Special and double-byte characters do not display correctly as binary. Your choice depends on your preference for entity names or character references.
XML without XHTML DTD/Schema	charref; use utf-8 only if your site uses UTF-8 consistently	XML supports only a very limited set of entity names unless the XHTML (or other) DTD is provided.
Not sure	charref	charref works with both UTF-8 encoding and XML parsers. It also gives the best results in Netscape. If special characters always appear as West European letters instead of the proper language, try latin.

Character Encoding Checklist

This section provides a checklist for setting the correct character encoding for content authored in **eWebEditPro+XML**.

1. Ensure the Web page with the **eWebEditPro+XML** editor has the proper charset specified in a meta tag.

```
<meta http-equiv="Content-Type" content="text/html; charset=big5">
```

2. Ensure the **eWebEditPro+XML** charset parameter matches the charset used in the meta tag. With **eWebEditPro+XML** 2.5 or higher and IE, this is done for you. Otherwise, if using a non-Western European characters, it is a good idea to explicitly set the charset parameter. You can do this in the ewebeditprodefaults.js file or on the page with the editor in JavaScript prior to creating the editor.

```
eWebEditPro.parameters.charset = "big5";
```

3. Ensure the charencode attribute in the **eWebEditPro+XML** configuration data is set properly. For information on choosing a value, see ["Configuring for Extended and Special Characters" on page 419](#).

As a general rule, use utf-8 if your charset is UTF-8, and use charref or binary if using non-Western European languages. 'binary' conforms to the charset of

the page with the editor. 'charref' is 7-bit ASCII and works with any charset and database, but may require special consideration when searching (see below).

```
<!-- values for charencode: utf-8, binary, entityname, charref, special, latin -->
<clean charencode="binary" ...>
```

If you are storing content in a database, ensure the database supports the encoding used. Some databases do not support Unicode (double byte characters). If it does not, you may wish to use UTF-8 or ASCII (with Unicode characters expressed as character references). See also How to store unicode characters so they are searchable.

Ensure the Web page that displays the content has the proper charset specified in a meta tag.

```
<meta http-equiv="Content-Type" content="text/html; charset=big5">
```

The browser should automatically select the proper encoding, but if it does not, correct the encoding. On IE 5, it is set from the View Encoding menu.

UTF-8

UTF-8 is not Unicode. Instead, it is a byte-stream representation of Unicode characters, which are always two bytes long. 7-bit ASCII characters are compatible with UTF-8 (that is, the same in UTF-8). UTF-8 characters may be one to three bytes long.

How to Store Unicode Characters So They Are Searchable

This section describes how to store unicode characters, such as Arabic or Japanese characters, so they can be searched when doing site searches.

NOTE	Your database must support Unicode (double-byte) characters. Unicode is not the same as UTF-8.
-------------	--

1. Make sure that your site specifies the proper character set for the characters you are trying to display.
2. In the config.xml file, set the charencode attribute of the clean element to "BINARY".
3. In the ewebeditprodefaults.js file, set the this.charset variable to the character set you want to use. Alternatively, you can use eWebEditPro.parameters.charset to specify it on the page.

For example, if you want to store Japanese characters, the clean element in the config.xml file would look like

```
<clean charencode="binary"....>
```

and the this.charset in the ewebeditprodefaults.js file will look like

```
this.charset="shift-jis";
```

or you can insert eWebEditPro.parameters.charset="shift-jis" on the page that calls the editor.

References

- Character entity references in HTML 4 (<http://www.w3.org/TR/REC-html40/sgml/entities.html>)
- The ISO 8859 Alphabet Soup (<http://czyborra.com/Charsets/iso8859.html>)
- Dan's Web Tips: Characters and Fonts (<http://www.dantobias.com/webtips/char.html>)
- W3C Internationalization/localization (<http://www.w3.org/International/Overview.html>)
- Character sets supported by popular Web applications (<http://www.w3.org/International/O-charset-list.html>)

Implementing a Web Site that Uses UTF-8 Encoding

UTF-8 is a byte stream encoding scheme that converts each double-byte Unicode character to one, two or three bytes.

For example, the letter a has an ASCII value of 97 (61 hex). It maps unchanged in UTF-8 to a single byte with a value of 97.

The single quote character (') has an ASCII value of 231 (E7 hex). It converts to two UTF-8 bytes: 195 167 (C3 A7 hex).

The Japanese single quote character (‘) has a Unicode value of 27231 (6A5F hex). It converts to three UTF-8 bytes: 230 169 159 (E6 A9 9F hex).

See Also:

- “[Encoding Special Characters](#)” on page 417
- “[Implementing UTF-8](#)” on page 427

Implementing UTF-8

To implement UTF-8, follow these points.

- All Web pages that include the editor or that display the content must set the charset to UTF-8.

```
<head>
meta http-equiv=Content-Type content="text/html; charset=utf-8">
...
</head>
```

- If you are using a database, ensure that it can accept UTF-8 or Unicode characters.
- Set the configuration data to produce characters for UTF-8.

```
<clean charencode="utf-8" ...>
```

(For more information, see “[charencode Attribute](#)” on page 419.)

- Load UTF-8 encoded content into the hidden field for the editor. How you load this content varies according to your server platform and environment.

```
<input type=hidden name="MyContent1" value="Content that is UTF-8 and HTML encoded">
```

NOTE

You may not be able to use standard HTML encoding functions, such as `HTMLEncode()` in ASP.

WARNING!

Content stored in JavaScript string variables and in the **eWebEditPro+XML** ActiveX control is stored as Unicode (double-byte) characters. When a

browser reads the value of the hidden field, the browser converts the UTF-8 byte stream to a Unicode string for JavaScript. Similarly, when a form is posted to the server, the browser converts content stored in the hidden field to UTF-8.

Tips

- If the UTF-8 byte stream is treated as a Unicode string, the special characters are corrupted and appear as two or three characters.
- If a Unicode string is interpreted as UTF-8, special characters are corrupted, and the number of characters is reduced, thereby eliminating some characters whether or not they are special.
- ASCII characters (A-Z, a-z, 0-9, etc.) always appear correctly because they have the same value in Unicode and UTF-8.

Setting the charset Parameter

If you are retrieving the entire document from the editor, set the charset parameter to **utf-8**. If you are retrieving only the body contents, you may still set the charset parameter.

To set the charset parameter to **utf-8**, update the ewebeditprodefaults.js file so that charset is set to **utf-8**. Or, you can use JavaScript to modify eWebEditPro.parameters on the page using this code.

```
eWebEditPro.parameters.editor.charset = "utf-8";
```

Browser Support for UTF-8

In order for the browser to support UTF-8, the following conditions must exist.

- The browser displaying the editor must support UTF-8.
 - Microsoft provides language add-ons for Internet Explorer.
 - Because Netscape 4.7x may not display Asian characters on English Windows (they may appear as question marks '?'), you need a language-specific version of the operating system and/or browser.
 - Netscape 6 supports multiple languages.
- Ensure that the browser encoding uses UTF-8. Set the browser to unicode using the sequence of menu options indicated below.
 - Internet Explorer: **View > Encoding > Auto-Select or Unicode (UTF-8)**
 - Netscape 4.7: **View > Character Set > Unicode (UTF-8)**
 - Netscape 6: **View > Character Coding > Auto-Detect > Auto-Detect (All) or Unicode (UTF-8)**

For More Information about UTF-8

- UTF-8 (technical specification) - <http://www.ietf.org/rfc/rfc2279.txt>
- The ISO 8859 Alphabet Soup - <http://czyborra.com/charsets/iso8859.html>

- Dan's Web Tips: Characters and Fonts - <http://www.dantobias.com/webtips/char.html>

Style Sheets

A style sheet is a file (extension .css) that contains specifications for the visual elements of a Web page, such as heading sizes, fonts and margins. You use a style sheet to override default HTML values for these elements on a group of Web documents or an entire Web site.

Style sheets let you establish a set of style specifications and apply them to all pages. Assume, for example, that the default display for the <H3> tag is Times New Roman.

Heading 3 default

If you apply a style sheet, it might modify the <H3> tag, like this.

```
h3 {FONT-FAMILY: Arial; FONT-SIZE: 14pt; MARGIN: 12pt 0in 3pt}
```

The text follows the style sheet specifications, and looks like this.

Heading 3 default

As a result, a Web site containing thousands of pages and updated by scores of editors can have a consistent look.

A good Web site that explains style sheets is <http://www.w3schools.com/css/default.asp>.

This section explains the following topics relating to using style sheets with eWebEditPro+XML.

- Using Style Sheets to Standardize Formatting
- The Default Style Sheet
- Applying Style Sheets
- The BodyStyle Parameter
- Preserving Tags When Office Content is Pasted
- Saving Style Sheet Tags When Content is Saved
- Inserting span or div Tags
- Applying Two Style Classes to the Same Content
- Implementing Style Class Selectors

Using Style Sheets to Standardize Formatting

You can combine a style sheet with the toolbar configuration procedures (see “Defining the Toolbar” on page 230) to control the formatting of the content that users produce.

For example, you could remove from the toolbar the menu options that let users select font size, color and style. Then, in a style sheet, you would specify a font size, color and style. If you make these modifications, users can enter text but not change its size, color or style -- the style sheet has standardized those specifications.

NOTE The bodyStyle parameter also lets you apply style sheet attributes to the content. See “Property: bodyStyle” on page 159.

The Default Style Sheet

eWebEditPro+XML provides a default style sheet, ektnormal.css, that emulates the Word 2000 Normal.dot template. If you assign this style sheet in the configuration data, the Word 2000 default styles are applied to the content.

To do this, set the href attribute in the features > standard > style section of the configuration data to look like this.

```
<features>
  <standard autoclean="true" publish="xhtml">
    <style publishstyles="true" href="[eWebEditProPath]/ektnormal.css"/>
```

Changing the Default Style Sheet

To change the default style sheet, place the new style sheet into the folder that contains **eWebEditPro+XML** and replace the named style sheet in the configuration data (above in red).

For example, if your custom style sheet is named mystyles.css, the configuration data would look like this:

```
<style publishstyles="true" href="[eWebEditProPath]/mystyles.css"/>
```

Applying Style Sheets

You can create your own style sheet and apply it to the **eWebEditPro+XML** editor. There are three levels at which you can apply a style sheet.

- **the configuration data** - affects all editors that refer to it (see “The Configuration Data” on page 312).

NOTE If your **eWebEditPro+XML** pages refer to several config.xml files (for example, you have different files for different user groups), and you want all pages to use the same style sheet, assign the same style sheet in all of the configuration data.

- **a page** - affects only the editors on one page
- **a single occurrence of the editor** - affects only one instance of the editor

WARNING!

Depending on your settings, you probably also need to specify the style sheet when the content is published. If you use templates in your Web application (for example, a content management system), a reference to the style sheet is required. This is typically done using the link tag. For example: <link rel="stylesheet" type="text/css" href="/ewebeditpro5/xyz.css">.

Note that if a style sheet is specified in more than one location, the most local one takes precedence. For example, if a sheet is specified in all three locations listed above, the style sheet applied to the single occurrence of the editor would be used.

If the most local style sheet does not include a specification for a certain tag, the browser will display its default for that tag -- it does not look in higher level style sheets for that tag's specifications.

Specifying a Style Sheet in the Configuration Data

You assign a style sheet using the `style` tag of the configuration data. To implement a style sheet in the configuration data, follow these steps.

NOTE

You can also apply, list and disable style sheets using ActiveX methods. For more information, see "For details on the properties, methods and events, see "eWebEditPro ActiveX Control Object" on page 13." on page 311.

1. Create your style sheet file (for example, xyz.css).
2. Open the config.xml file in the directory where you installed eWebEditPro+XML.
3. Move to the `style` tag, located within the `features > standard` section of the configuration data.

```
<features>
  <standard autoclean="true" publish="xhtml">
    <style publishstyles="true" href="[eWebEditProPath]/ektnormal.css"/>
```

Note that `[eWebEditProPath]` refers to the eWebEditProPath variable in the ewebeditpro.js file. If your style sheet resides in a different directory, replace `[eWebEditProPath]` with the directory pathway.

4. Change the `href` attribute in the `style` command so that it refers to your style sheet.

```
<style publishstyles="false" href="/yourpath/xyz.css"/>
```

5. Set `publishstyles` to `false`.

```
style publishstyles="false" href="/yourpath/xyz.css"/>
```

Adding a Style Sheet to a Single Page

1. Open the page to which you want to add a style sheet.
2. Set the `styleSheet` parameter by adding JavaScript to the page before the editor is created.

```
<script language="JavaScript1.2">
<!--
    eWebEditPro.parameters.styleSheet = "/yourpath/xyz.css";
// --
</script>
<!-- code to place the editor on the page goes here -->
```

Dynamically Changing a Style Sheet for a Single Instance of the Editor

1. Open the page to which you want to add a style sheet.
2. Add the following JavaScript function below the page's head tag.

```
<script language="JavaScript">
function setStyleSheet(strEditorName, strCSS)
{
    eWebEditPro[strEditorName].setProperty("StyleSheet", strCSS);
}
</script>
```

Replace `strEditorName` with the name of the editor, and `strCSS` with the name of the style sheet.

3. On the page where you create **eWebEditPro+XML**, set the `onready` event to call the `setStyleSheet` function.

For example,

```
<script language="JavaScript">
eWebEditPro.onready = "setStyleSheet(eWebEditPro.event.srcName, '/yourpath/xyz.css') ";
</script>
```

Tip: You can set the `StyleSheet` property to change the style sheet after the editor loads. For example, you might want to change the style sheet when the user picks from a list of styles that you provide.

The BodyStyle Parameter

The `BodyStyle` parameter also affects all editors, or an instance of the editor. If the `body` style parameter is set, it takes precedence over a style sheet. The `parameter` applies the style to the `style` attribute of the `body` tag.

For more information, see “[Property: bodyStyle](#)” on page 159.

Preserving Tags When Office Content is Pasted

Within the configuration data, the `style` tag has `preservewordstyles` and `preservewordclasses` attributes that determine whether class and style attributes are preserved when Microsoft Office 2000 or later content is pasted into the editor.

If you set these attributes to **true**, class and style attributes are preserved when pasting Word 2000 content. If set to **false**, the class and style attributes are removed.

Below is an example of how to implement this feature within the configuration data.

```

<features>
</external>
<standard autoclean="true" publish="xhtml">
<style preservewordstyles="true" preservewordclasses="true"/>

```

Saving Style Sheet Tags When Content is Saved

Within the configuration data, the style tag has a `publishstyles` attribute that determines whether the style sheet specifications for each tag are inserted into the file *when the content is saved*.

```

<features>
</external>
<standard autoclean="true" publish="xhtml">
<style publishstyles="true"/>

```

Below is an example of the html text of a saved line when `publishstyles` is set to **true**.

```
<p align="center" style="BOTTOM: 0px; FILTER:; FONT-FAMILY: 'Times New Roman';
FONT-SIZE: 12pt; MARGIN: 0in 0in 0pt"> VARs benefits and features</p>
```

Here is the same line when `publishstyles` is set to **false**.

```
<p> VARs benefits and features </p>
```

Setting Publishstyles to True

Set `publishstyles` to **true** to make sure that the formatting specifications remain with the content after it is saved.

Setting Publishstyles to False

Set `publishstyles` to **false** to maintain control of the styles for an entire Web site. In this case, you would not want to insert style sheet specifications for each tag. Instead, your style specifications would be taken from the style sheet specified in the display page's head tags or, if you are using a content management system, from the template file.

Another advantage of setting `publishstyles` to **false** is that it greatly reduces the size of the html page (as you can see from the example above).

Inserting span or div Tags

The `wrapstylewithdiv` attribute determines what to do when a user applies a generic style class to text surrounded by blocking tags. Set the attribute to **true** to wrap such text with `<div>` tags. To wrap this text with `` tags, set the attribute to **false**.

For example, assume you have this content.

RC International is dedicated to the RC racing enthusiasts! We eat, work, play, and live RC racing. In three short years RC International has become one of the leading manufacturers of RC racing and flying vehicles. Our dedication to the sport, and the enthusiasts who play it, has endeared our products to the RC community.

Also, assume that you want to apply the following generic style class to content that crosses paragraphs:

```
.uppercase
{
text-transform: uppercase;
}
```

If you set the attribute to "true and apply uppercase to the following text (which crosses paragraphs) "We eat, work, play, and live RC racing.

In three short years RC International has become one of the leading manufacturers of RC racing and flying vehicles.", the HTML source looks like this:

```
<p>RC International is dedicated to the RC racing enthusiasts!</p>

<div class="uppercase">
<p>We eat, work, play, and live RC racing. </p>
<p>In three short years RC International has become one of the leading manufacturers of RC racing and flying vehicles.</p>
</div>
<p>Our dedication to the sport, and the enthusiasts who play it, has endeared our products to the RC community.</p>
```

Because `<div>` tags add `<p>` tags, in WYSIWYG mode, the text looks like this:

RC International is dedicated to the RC racing enthusiasts!

WE EAT, WORK, PLAY, AND LIVE RC RACING.

IN THREE SHORT YEARS RC INTERNATIONAL HAS BECOME ONE OF THE LEADING MANUFACTURERS OF RC RACING AND FLYING VEHICLES.

Our dedication to the sport, and the enthusiasts who play it, has endeared our products to the RC community.

As you can see, the new `<p>` tags change the paragraph formatting. To avoid this problem, set the `wrapstylewithdiv` attribute to "false". If you do, the editor wraps the selected text with `` tags within the blocking tags. `` tags do not affect the paragraph formatting.

Here is the HTML source when the attribute is set to false.

```
<p>RC International is dedicated to the RC racing enthusiasts!<span class="uppercase">We eat,  
work, play, and live RC racing.</span></p>  
<p><span class="uppercase">In three short years RC International has become one of the leading  
manufacturers of RC racing and flying vehicles.</span>
```

Our dedication to the sport, and the enthusiasts who play it, has endeared our products to the RC community.</p>

In WYSIWYG mode, the text looks like this:

RC International is dedicated to the RC racing enthusiasts! WE EAT, WORK, PLAY, AND LIVE RC RACING.

IN THREE SHORT YEARS RC INTERNATIONAL HAS BECOME ONE OF THE LEADING MANUFACTURERS OF RC RACING AND FLYING VEHICLES. Our dedication to the sport, and the enthusiasts who play it, has endeared our products to the RC community.

Applying Two Style Classes to the Same Content

When a user applies a new style class to content to which a style class is already applied, it is not obvious what the editor should do:

- Should it replace the original style class with the new?
- Should it add the new style class around the original?

The `equivClass` attribute of the configuration data lets you control the editor's behavior when a user applies a style class to content to which another style class is already applied.

Location of `equivClass` Attribute

The `equivClass` attribute is located in the **features > style** tag of the configuration data.

```
<style publishstyles="false" href="[eWebEditProPath]/ektnormal.css" equivClass="strict"  
wrapstylewithdiv="false" preservewordstyles="true">
```

How the Editor Determines if Two Classes Are Equivalent

When a user applies a new style class to content to which a style class is already applied, the editor

1. compares the properties of the original and new style classes, and
2. refers to the `equivClass` attribute to determine which style properties should apply to the content

After comparing the original and new style classes, the editor determines whether the two style classes are "equivalent".

NOTE

Property values are ignored - only property names are considered.

You control how the editor defines "equivalent" through the `equivClass` attribute. This attribute has three values.

equivClass attribute value	The two classes are equivalent
<code>strict</code>	if they have exactly the same properties
<code>loose</code>	if they share at least one property. See Also: " Forcing Two Classes to be Equivalent " on page 438
<code>all</code>	regardless of similarity among properties

The result of this comparison is that the two style classes (original and new) are determined **equivalent** or **not equivalent**.

New Class is Equivalent to Original Class

If the style classes are equivalent, the editor replaces the original class with new class. For example

before

```
<span class="original"> Hello World </span>
```

after

```
<span class="new"> Hello World </span>
```

New Class is not Equivalent to Original Class

If two style classes are not equivalent, the editor adds the new style class around original style class. For example

before

```
<span class="original"> Hello World </span>
```

after

```
<span class="new"><span class="original"> Hello World </span></span>
```

As a result,

- if a property occurs in both classes, the original class property is applied because it is closer to the content
- if a property occurs in only one class, it is applied to the content

For example, here are two style classes:

```
.original
```

```

{
    font-size: small;
    color : red;
}

.new
{
    font-size: large;
    background-color : Gray;
}

```

Because the `font-size` attribute occurs in both styles and the `.original` style is closer to the content, the `.original` size (small) is used. On the other hand, `color` only occurs in the `.original` style class, and `background-color` only occurs in the `.new` style class, so both are applied to the content.

Forcing Two Classes to be Equivalent

You can force two classes to be equivalent even if they have no common properties. To do this, add the Ektron-specific style class property, `equivClass`, to each style class that you want to be equivalent. For example,

```

.red
{
    equivClass: Group1;
    color : red;
}

.backcolor
{
    equivClass: Group1;
    background-color : Gray;
}

```

In this example, the two style classes, `.red` and `.backcolor`, are considered equivalent because they have the same value for the `equivClass` property.

Tips for Using this Feature

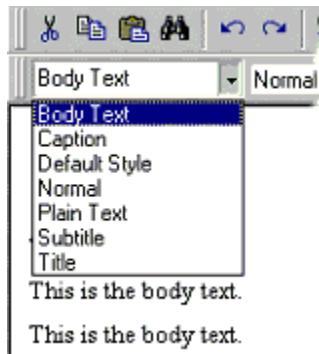
If you want to	Set the <code>equivClass</code> attribute to
Have the new style class always replace the original	all
Have the new style class replace the original if at least one of its properties matches at least one of the original style class' properties	loose

If you want to	Set the equivClass attribute to
Have the new style class replace the original if all of its properties match the original style class' properties. Otherwise, the new style class is applied around the original.	strict

Implementing Style Class Selectors

You can add to the toolbar a dropdown list (`cmdselstyle`) that lets users choose a style class and apply it to selected text.

See Also: ["Adding a Dropdown List" on page 238](#)

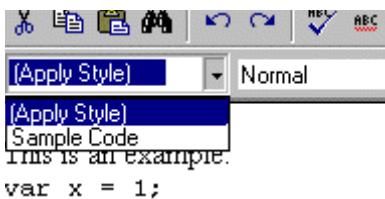


The styles appear in the order in which they are entered into the style sheet assigned to the editor.

Example of Using Style Class Selectors

As an example of using style class selectors, assume that your Web site features text that is sample programming code. The Webmaster would open the organization's style sheet, create a style class called "Sample Code" and assign appropriate formatting specifications to it (such as `font-size:9.0pt; font-family:"Courier New"`).

Then, when a user types sample programming code into the editor, he could select the code, click the dropdown list, and select **Sample Code** from the list (see illustration).



The HTML code for this line would look like this:

```
<p class="code">var 1 = x;</p>
```

If a user wants to later remove a style class, he would select the text and press the Remove Style button ().

Types of Style Classes

There are two types of style classes.

Type	Example	Can be applied to
tag specific	p.box { border: solid 2px red }	Only HTML tags specified in the definition. The example style can only be applied to text surrounded by <p> tags. Affects entire paragraph.
generic	.highlight { background-color: yellow; }	Selected text using or <DIV> tags, regardless of tags surrounding the text. Affects selected text only. <i>See Also: "Inserting span or div Tags" on page 434</i>

The following sample code illustrates both kinds of style classes.

```
<p class="box">IMPORTANT: read this <span class="highlight">highlighted</span> word</p>
```

Determining Which Style Classes Appear in the Dropdown List

Styles appear on the list only if they satisfy these criteria:

- the style's `visible` attribute is not set to `false` (`visible` is not a standard attribute, and is only present if someone adds it to the style sheet or you use eWebEditPro+XML's default style sheet, `ektnormal.css`)
- the style does not have a tag specifier or the tag specifier matches the current tag. For example, if selected text is surrounded by `<p>` tags, and the style class has a tag specifier of `a`, the style does not appear on the list. (For more information, see "Types of Style Classes" on page 440.)

- the style must have a class specifier. For example, if the selected text is surrounded by <p> tags, p.normal {} and .highlight {} appear on the list, but p {} would not appear because it has no class specifier.

Determining the Names in the Dropdown List

By default, a style class' name without the tag prefix appears in the dropdown list. For example, the style class p.highlight appears as **highlight**.

If you want to change the name, use the `caption` attribute within the style class definition. For example, to have the `p.highlight` class appear as **yellow background** in the dropdown list, enter the following into the style sheet definition:

```
p.highlight
{
  caption : yellow background;
  border : thin solid Green;
}
```

Translating Style Class Names

You can translate the dropdown list so that non-English speaking users see it in their native language. To accomplish this, assign a `localeRef` attribute and code to a style class. For example

```
.code {
  localeRef:cssCode;
```

Then, translate the code to a foreign term in the appropriate `locale.xml` file. When the editor displays the list, it displays the style names from the localization file. (For more information, see “[Translating Button Captions and Tool Tips](#)” on page 244.)

For example, assume that your users speak French, so you would modify the `locale040cb.xml` localization file. Also, assume that the style “Sample Code” translates into “Code d'échantillon” in French.

Here is an example of a *standard* style sheet specification. (The red is added for emphasis.)

```
.code {
  caption:Sample Code;
  margin:0in;
  font-size:10.0pt;
  font-family:"Courier New";}
```

Here is a style sheet specification with a reference to a `localeref`.

```
.code {
  localeRef:cssCode;
  margin:0in;
  font-size:10.0pt;
  font-family:"Courier New";}
```

Here is how to update the `locale040cb.xml` localization file so that it displays “Code d'échantillon” on the dropdown list.

```
<cssCode>Code d'échantillon</cssCode>
```

Suppressing Styles from the Dropdown List

If you want to suppress styles from the dropdown list, add `visible:false` to the style class's definition in the style sheet. For example

```
.code {
  visible:false;
  margin:0in;
  font-size:10.0pt;
  font-family:"Courier New"; }
```

Style Classes and Matching Attributes

Some style classes have attributes that match attributes of other style classes. Here is an example (both style classes have a font style attribute.)

```
.normal
{font-style: normal;}
.italic
{font-style: italic; }
```

This section describes how **eWebEditPro+XML** handles matching attributes when a style class is applied to Web content, and then another style class is applied to the same content.

To understand how **eWebEditPro+XML** reacts when another style class is applied, the following table describes the three attribute match possibilities.

Two style classes have	Example
the same attributes	<pre>.normal {font-style: normal;} .italic {font-style: italic; }</pre>
some same attributes and some different attributes	<pre>.normal {font-style: normal;} .italic_overline {font-style: italic; text-decoration : overline; }</pre>
different attributes	<pre>.normal {font-style: normal;} .overline {text-decoration: overline; }</pre>

How **eWebEditPro+XML** handles each possibility is described below.

Style Classes Have Same Attributes

If a user applies one style class and then applies another with the same attributes, the second style class replaces the first.

Style Classes in this Example

```
.normal
{font-style: normal;}
```

```
.italic
{font-style: italic; }
```

Before

HTML	WYSIWYG
<P>This is initial content.</P>	This is initial content.

After

HTML	WYSIWYG
<P>This is initial content.</P>	This is <i>initial</i> content. (.italic style class replaces .normal)

Style Classes Have Some Similar and Some Different Attributes

If a user applies one style class and then another with some of the same and some different attributes, the second class' same attributes override the first class' matching attributes.

Style Classes in this Example

```
.normal
{font-style: normal; }
.italic_overline
{font-style: italic;
text-decoration: overline; }
```

Before

HTML	WYSIWYG
<P>This is initial content.</P>	This is initial content.

After

HTML	WYSIWYG
<P>This is initial content.</P>	This is <u><i>initial</i></u> content. (Because the styles are not exact match, both SPAN tags remain in HTML. Text is italic because second SPAN tag changes font style.)

Style Classes Have Different Attributes

If a user applies one style class and then another with different attributes, the first class' attribute remains, because the second class does not have that attribute.

Style Classes in this Example

```
.normal
{font-style: normal;}
.overline
{text-decoration: overline;}
```

Before

HTML	WYSIWYG
<P>This is initial content.</P>	This is initial content.

After

HTML	WYSIWYG
<P>This is initial content.</P>	This is <u>initial</u> content. (Text is normal because second SPAN tag does not have font-style attribute.)

Managing Hyperlink Dialogs

eWebEditPro+XML's standard toolbar features three buttons that let users manage hyperlinks within their content.

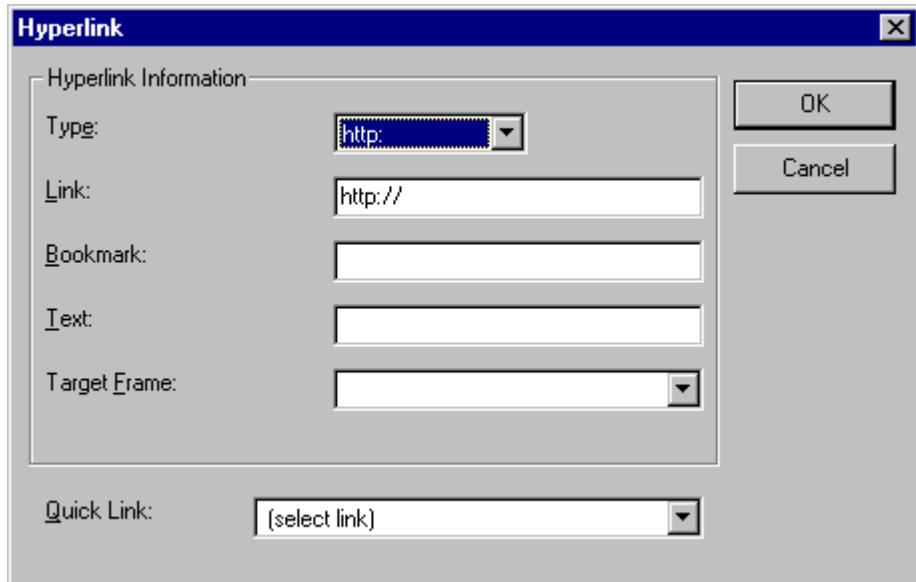
- The *Edit Hyperlink* (cmdhyperlink) toolbar button (- The *Remove Hyperlink* (cmdunlink) toolbar button (img alt="Remove Hyperlink icon" data-bbox="795 225 815 245") lets users remove a hyperlink
- The *New Hyperlink* (jshyperlink) toolbar button (img alt="New Hyperlink icon" data-bbox="795 265 815 285") lets users add a hyperlink to their Web content

NOTE

By default, this button does not appear on the toolbar. If you would like to use it, you must add it.

Customizing Dropdown Lists in the Hyperlink Dialog Box

This section explains how to customize the Hyperlink dialog box (illustrated below).



Specifically, the section explains how to customize the

- values that appear in dropdown lists (see “Customizing the Lists of the Hyperlink Dialog Box” on page 446)
- default values for most fields (see “Specifying Default Values for the Insert Hyperlink Dialog” on page 452)

Customizing the Lists of the Hyperlink Dialog Box

You edit the Hyperlink dialog box's lists within the configuration data, under command name="cmdhyperlink". By default, these lists are not part of the configuration data. As a result, you must first add each list that you want to customize to the configuration data.

After you add a list to the configuration data, customize the list by

- adding or deleting list items (for example, deleting the mailto protocol)
- changing attribute values (for example, to make the list of protocol types disappear from the Hyperlink dialog box, change type's visible attribute to "false")

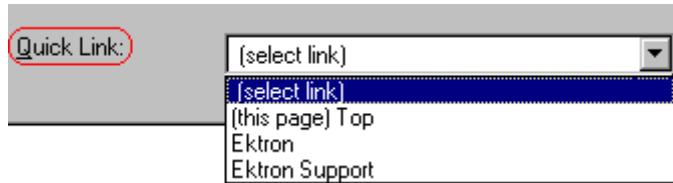
The Hyperlink dialog box's fields whose values you can determine are

- **Quick Link** (see “[Quick Link List](#)” on page 446)
- **Type** (see “[Type List](#)” on page 448)
- **Target Frame** (see “[Target Frame List](#)” on page 450)

Quick Link List

Populates the “Quick Link” list, the list of URLs or other Web destinations to which users will typically want to create jumps.

Illustration



Example

```
<command name="cmdhyperlink" >
  <image key="hyperlink"/>
  <caption localeEef="cmdHyp"/>
  <tooltiptext localeRef="cmdHyp"/>
  <selections name="quicklink" visible="true" bookmarks="true" listtop="false">
    <listchoice href="http://www.ektron.com" target="_blank">Ektron Home Page</listchoice>
  </selections>
</command>
```

Directions for Updating

1. Open config.xml.
2. Find the section of the file that begins with cmdhyperlink.

3. If the group of listchoice elements shown in the example above does not appear under the cmdhyperlink command, copy and paste the sample selections list (above) into config.xml under the cmdhyperlink command.
4. To *add* a quick link, copy and paste the line `<listchoice href="http://www.ektron.com" target="_blank">Ektron Home Page</listchoice>` within the selections tags. Then, replace the copied values (in this example, "http://www.ektron.com" and Ektron Home Page) with new values.
To *remove* a quick link, delete the entire line on which it appears.

Selection Elements

Element Attribute	Value(s)	Description
name	quicklink	The name of this dropdown list.
visible	true (default)	The Quick Links list is visible.
	false	The Quick Links list is not visible.
bookmarks	true (default)	Bookmarks on this page appear in the Quick Links list.
	false	Bookmarks on this page do not appear in the Quick Links list.
listtop	true (default)	The "Top" bookmark appears in the Quick Links list.
	false	The "Top" bookmark does not appear in the Quick Links list.
listchoice/ href		The URL of a destination to which the user clicking this link is brought.
listchoice/ target	Any valid frame name or one of the following special names: _blank, _self, _parent, _top	Target window (frame name). If you specify a target frame, and the user is allowed to select a target frame (at the Target Frame field), the user's choice will override this value.
listchoice/ localeRef	refID	A code to translate this element within the localization files (typically not used).
listchoice/ <display text>		Text that describes the destination in the Quick Links list.

Type List

Determines which protocols a user can assign to a link.

Illustration**Example**

```

<command name="cmdhyperlink" >
  <image key="hyperlink" />
  <caption localeRef="cmdHyp" />
  <tooltiptext localeRef="cmdHyp" />
  <selections name="type" enabled="false" visible="true">
    <listchoice data="0">file:</listchoice>
    <listchoice data="0">ftp:</listchoice>
    <listchoice data="0">gopher:</listchoice>
    <listchoice data="0" default="true">http:</listchoice>
    <listchoice data="0">https:</listchoice>
    <listchoice data="1">JavaScript:</listchoice>
    <listchoice data="1">mailto:</listchoice>
    <listchoice data="1">news:</listchoice>
    <listchoice data="0">telnet:</listchoice>
    <listchoice data="0">wais:</listchoice>
  </selections>
</command>

```

Directions for Updating

1. Open Config.xml.
2. Find the section of the file that begins with cmdhyperlink.
3. If the group of listchoice elements shown above does not appear under the cmdhyperlink command, copy and paste the sample selections list (above) into config.xml under the cmdhyperlink command.
4. To add a protocol, copy and paste the line `<listchoice data="0">file:</listchoice>` within the selections tags. Then, replace the copied value (in this example, file:) with the new value.

To remove a protocol, delete the entire line on which it appears.

Selection Elements

Element Attribute	Value(s)	Description
name	type	The name of this dropdown list.
enabled	true (default)	User can select from the list.
	false	User cannot select from the list; selections are grayed out.
visible	true (default)	List is visible.
	false	List is not visible.
listchoice/ data	0	Protocol requires double slash marks (//). For example, <code>http://www.yoursite.com</code> .
	1	Protocol does not require double slash marks (//). For example, <code>mailto:you@email.com</code> .
listchoice/ default	true	This choice is the default type. <u>Note:</u> <code>http:</code> is the default type if no value is specified.
	false (default)	This choice is not the default type.
listchoice/ text value	Any valid protocol (including the colon). Typically, one of the following: <code>file:</code> , <code>ftp:</code> , <code>gopher:</code> , <code>http:</code> , <code>https:</code> , <code>JavaScript:</code> , <code>mailto:</code> , <code>news:</code> , <code>telnet:</code> , <code>wais:</code>	The internet protocols from which the user can choose.

Target Frame List

Determines target window choices.

Illustration**Example**

```
<command name="cmdhyperlink" >
  <image key="hyperlink"/>
  <caption localeRef="cmdHyp"/>
  <tooltiptext localeRef="cmdHyp"/>
  <selections name="target" enabled="true" visible="false">
    <listchoice value="main">Main Frame</listchoice>
    <listchoice value="_blank" localeRef="hypTargB"></listchoice>
    <listchoice value="_self" localeRef="hypTargS" default="true"></listchoice>
    <listchoice value="_parent" localeRef="hypTargP"></listchoice>
    <listchoice value="_top" localeRef="hypTargT"></listchoice>
  </selections>
</command>
```

Directions for Updating

1. Open config.xml.
2. Find the section of the file that begins with cmdhyperlink.
3. If the group of listchoice elements shown in the example above does not appear under the cmdhyperlink command, copy and paste the sample selections list (above) into config.xml under the cmdhyperlink command.
4. To add a target window, copy and paste the line `<listchoice value="_blank" localeRef="hypTargB"></listchoice>` within the selections tags. Then, replace the copied value (in this example, _blank) and localeRef with new values.

To remove a target window choice, delete the entire line on which it appears.

Selection Elements

Element Attribute	Value(s)	Description
name	target	The name of this dropdown list.
enabled	true (default)	User can select from the Target Frame list.
	false	User cannot select from the Target Frame list; selections are grayed out.
visible	true (default)	Target Frame list is visible.
	false	Target Frame list is not visible.
listchoice/value	Any valid frame name or one of the following special names: _blank, _self, _parent, _top	The list of target window types from which the user can choose.
listchoice/localeRef	refID	A code to translate this element within the localization files.
listchoice/default	true	This choice is the default type.
	false (default)	This choice is not the default type.
listchoice/<display text>		Text to appear in the target list if no localeRef is found.

Specifying Default Values for the Insert Hyperlink Dialog

You can customize the default values that appear in the Insert Hyperlink dialog box. To do this, enter a text data argument of HTML hyperlink (that is, <A> tag) attributes when sending the command in JavaScript.

For example:

```
var strAttrs = "type='video/mpeg' href='ski.mpeg' text='Learn to Ski'";
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdhyperlink", strAttrs, 0);
```

If you do, the Insert Hyperlink dialog will have the default values specified in the attributes string. You can also specify attributes that do not appear in the dialog.

The following table explains how to set a default value for each field in the Insert Hyperlink dialog.

Field	How to Set Default Value
Type	Taken from the href attribute. For example, <code>href='ftp://domain.com/'</code> illustrates the type “ftp”.
Link	The href attribute without the protocol (Type) or bookmark
Bookmark	Taken from the href attribute. For example, <code>href='http://domain.com/file.htm#bookmark'</code>
Text	Use the 'text' pseudo attribute. For example, <code>text='Learn to Ski'</code>
Target Frame	The target attribute. For example, <code>target='_blank'</code>
Quick Link	Cannot set default

Entering the Sample Code

Enter the sample code in a `customevents.js` file, in a `onexeccommand` handler function (for details, see “[Creating a Custom Command](#)” on page 215). The command is executed when the user selects it from a custom dropdown list or presses a custom button.

To learn how to create a custom dropdown list, see “[Creating a Popup Menu](#)” on page 245.

To learn how to create a custom button, see “[Creating a Custom Command](#)” on page 215.

Editing the New HyperLink Dialog Box

The New Hyperlink toolbar button () lets users quickly add a hyperlink to their Web content. The hyperlink command, `jshyperlink`, resides in the external section of the configuration data.

To add a hyperlink, the user selects text, clicks the New Hyperlink button, and selects a “Quick Link” (the name assigned to a URL) from a drop down menu.



By default, the **Quick Links** field has three values:

- Ektron (<http://www.ektron.com>)
- Ektron Products (<http://www.ektron.com/products>)
- Ektron eWebEditPro (<http://www.ektron.com/ewebeditpro>)

Editing Quick Links

To edit the list of Quick Links from which a user can select, follow these steps.

1. Using your favorite program editor, open hyperlinkpopup.htm. This file is in your server's **eWebEditPro+XML** installation directory, typically **c:\inetpub\wwwroot\ewebeditpro5**.
2. Move to the section of the file that begins with **<td>Quick Links </td>**. That section looks like this.

```
<td>Quick Links </td>
<td>
<!---- This should be Dynamically created. ---->
<select name="hyperlinklist" size="1" onchange="movelink()">
    <option value=""></option>
    <option value="http://www.ektron.com">Ektron</option>
    <option value="http://www.ektron.com/products">Ektron Products</option>
    <option value="http://www.ektron.com/ewebeditpro">Ektron eWebEditPro</option>
</select>
</td>
```

Removing Quick Links

To remove any Quick Link, delete the entire line on which it appears.

Adding Quick Links

To add a Quick Link, follow these steps.

1. Copy and paste the line `<option value=""></option>` within the `select` tags.
2. Within the quotes (""), enter the URL that you want users to select as a Quick Link.
3. Following the greater than sign (>) after the quotes, enter the text that will be inserted into the Web content to identify the hyperlink.

For example, to provide a Quick Link to yahoo, the line would look like this.

```
<option value="http://www.yahoo.com">Yahoo</option>
```

Dynamically Creating the Quick Links File

You can dynamically create the Quick Links file, hyperlinkpopup.htm, and populate the list of Quick Links from a database. The hyperlinkpopup.htm file is specified within the ewebeditproevents.js file.

Managing Images

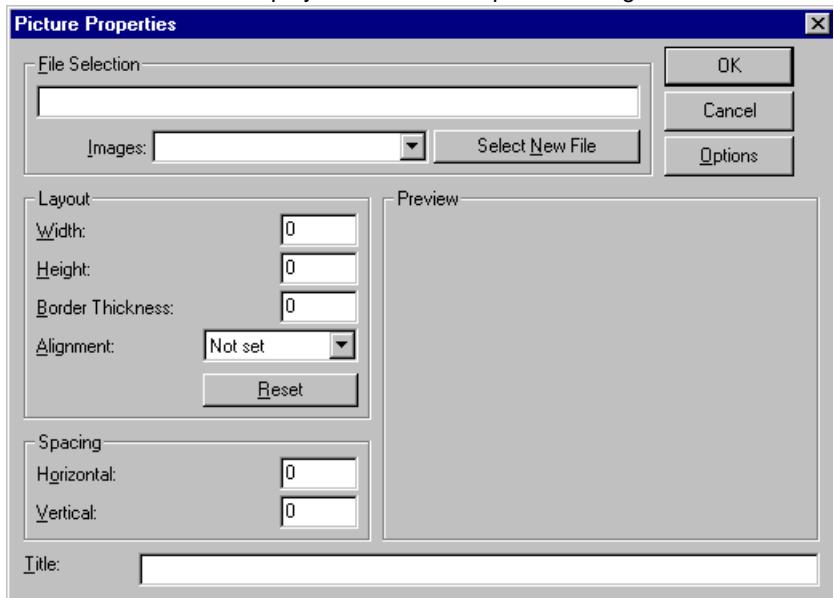
Typically, users insert images into content while editing within **eWebEditPro+XML**. How those images are uploaded to the server is described in "How Image Selection Works" on page 455.

However, if a user is editing within another application (such as Microsoft Word), the user can insert images within the other application, paste the content into **eWebEditPro+XML**, and upload those images to the server. This process is described in "Automatic Upload" on page 520.

How Image Selection Works

NOTE This section assumes that you have not edited the commands in the `mediafiles` feature of the configuration data.

1. The user clicks the Insert Image button () , which executes the configuration data's `cmdmfumedia` command.
2. The `cmdmfumedia` command calls the `eWebEditProMediaSelection` function in the `ewebeditpromedia.js` file. That function displays the Picture Properties dialog box.



3. The user clicks the **Select New File** button.
4. The editor checks the value of the `type` attribute of the `mediafiles` feature in the configuration data.
 - If you set the value to **FTP**, you need to set up image selection via FTP. (See "FTP File Upload" on page 472.)

- If you set the value to an HTML file pathway, that page is loaded. This option typically displays a screen that prompts the user to select an image. More details about this option are provided in “[Customizing the Alignment Field of the Picture Properties Dialog](#)” on page 457.
5. The Picture Properties dialog box reappears with the selected image. The user can change the image properties if desired.
 6. When the user clicks **OK**, the image is inserted into the content.

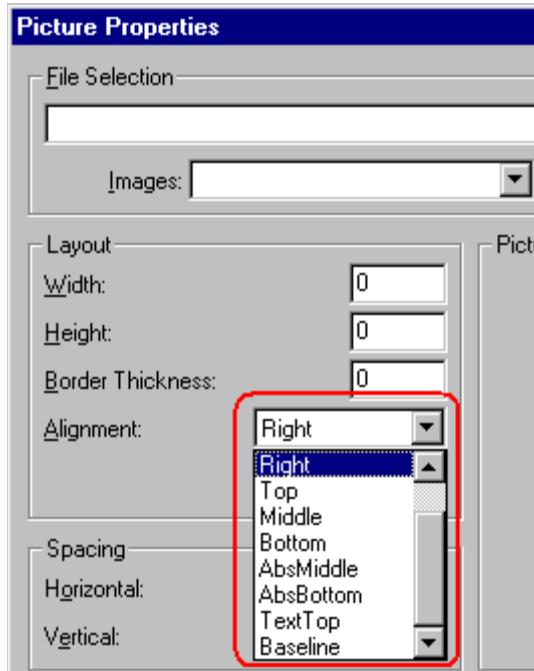
Organization of the Image Selection Documentation

The rest of this section describes the various aspects of the image selection feature.

This section	Describes
“Customizing the Alignment Field of the Picture Properties Dialog” on page 457	Modifying the Alignment field of the Picture Properties dialog box
“The ewebeditpromedia File” on page 296	Customizing the external media file selection process
“Examples of Implementing Image Selection” on page 459	How to create the image selection screen
“Implementing Image Upload” on page 472	How to implement media upload under different environments
“The Mediafiles Feature” on page 493	The elements of the mediafiles feature
“Manipulating Media File Methods and Properties” on page 486	The methods and properties of the Media File Object
“Programmatically Accessing Media File Properties” on page 488	Programmatically accessing the Media File Object’s properties
“Dynamically Selecting Upload Destinations” on page 513	Using scripting to change the image file upload location
“Automatic Upload” on page 520	Uploading images in content copied from another application

Customizing the Alignment Field of the Picture Properties Dialog

You can **modify** the list of possible responses to the **Alignment** field of the Picture Properties dialog box (illustrated below). You can also specify a **default response** or **remove** the field from the dialog.



Modifying Alignment Field Responses

To modify the list of possible responses to the **Alignment** field, enter a dropdown list of all possible values (illustrated below). Remove values that the user should not be able to select.

```
<mediaconfig enabled="true" allowededit="true">
  <selections name="alignment" visible="true">
    <listchoice value="" localeRef="picNS"/>
    <listchoice value="left" localeRef="picAliL"/>
    <listchoice value="right" default="true" localeRef="picAliR"/>
    <listchoice value="top" localeRef="picAliT"/>
    <listchoice value="middle" localeRef="picAliM"/>
    <listchoice value="bottom" localeRef="picAliB"/>
    <listchoice value="absmiddle" localeRef="picAliAM"/>
    <listchoice value="absbottom" localeRef="picAliAB"/>
    <listchoice value="texttop" localeRef="picAliTT"/>
    <listchoice value="baseline" localeRef="picAliBL"/>
  </selections>
</mediaconfig>
```

Note that

- The name of the selections list must be alignment .

- The text in the command attribute becomes the align value used. (The command in this list is not sent to the client scripting.)
- If no command value is given, when the user selects the option, no align attribute is assigned to the img tag.
- The #text is the description shown to the user. It does not need to match the text in the command attribute.
 - It can be translated using the localeRef attribute
 - If it is omitted or not translated, the text in the command attribute is used
- The list must be in either the cmdmfumedia command definition (shown below) or in the mediaconfig element (shown above). If the list appears in both locations, the cmdmfumedia command takes precedence.

```
<command name="cmdmfumedia" >
    <caption localeRef="cmdPic"/>
    <image key="picture"/>
    <tooltiptext localeRef="cmdMore"/>
    <selections name="alignment">
        <listchoice value="" localeRef="picNS" default="true">Not Set</listchoice>
        <listchoice value="left" localeRef="picAliL">My Left</listchoice>
        <listchoice value="right" localeRef="picAliR">My Right</listchoice>
        <listchoice value="top" localeRef="picAliT">Top</listchoice>
        <listchoice value="middle" localeRef="picAliM">Middle</listchoice>
        <listchoice value="bottom" localeRef="picAliB">bottom</listchoice>
        <listchoice value="absmiddle" localeRef="picAliAM">Absolute Middle</listchoice>
        <listchoice value="absbottom" localeRef="picAliAB">Absolute Bottom</listchoice>
        <listchoice value="texttop" localeRef="picAliTT">Text Top</listchoice>
        <listchoice value="baseline" localeRef="picAliBL">Base Line</listchoice>
    </selections>
</command>
```

Setting a Default Response for the Alignment Field

To specify a default response for the **Alignment** field, add the attribute default="true" to the default value. In the example below, right will be the default response for the **Alignment** field.

NOTE

The **selections** element must include at least one selection for the list to be valid. The visible attribute is only checked when there is a valid dropdown list.

```
<mediaconfig enabled="true" allowededit="true" >
    <selections name="alignment" visible="true">
        <listchoice value="" localeRef="picNS"/>
        <listchoice value="left" localeRef="picAliL"/>
        <listchoice value="right" default="true" localeRef="picAliR"/>
    <selections>
```

Removing the Alignment Field from the Picture Properties Dialog

To remove the **Alignment** field from the Picture Properties dialog box, set the dropdown list's visible attribute value to false, as illustrated below.

NOTE The selections element must include at least one selection for the list to be valid.

```
<mediaconfig enabled="true" allowededit="true">
    <selections name="alignment" visible="false">
        <listchoice value="left" localeRef="picAliL"/>
    </selections>
</mediaconfig>
```

Examples of Implementing Image Selection

This section provides four examples of how to create the image selection screen mentioned in Step 4 of “[How Image Selection Works](#)” on page 455. This table summarizes the examples.

Example	File Upload?	Upload protocol	Administrator restricts image?
1: No Restrictions, No Saving to Database	no	n/a	no
2: File Size Restriction, No Saving to Database	no	n/a	yes
3: FTP	determined by Web master	FTP	yes
4: Database Samples	yes - URL stored in database	HTTP	yes

Example 1: No Restrictions, No Saving to a Database

In this example, the user inserts an image from a remote directory. The image is not uploaded to a database, and no restrictions are imposed on the image.

To incorporate this version of image selection, follow these steps.

1. Within the ewebeditpro5 directory, create an .htm file, for example, imageselection.htm.
`<script language="JavaScript1.2" src="ewebeditpro.js">
<script>`
 (For more information, see “[Customizing the Alignment Field of the Picture Properties Dialog](#)” on page 457.)
2. Within the imageselection.htm file’s head tags, include the ewbeditpro.js file.
3. Still within the document’s head tags, create an insertfile function that calls the standard insertMediaFile function. (See “[Method: insertMediaFile](#)” on page 94.)

NOTE

In the following example, the editor name appears as `MyContent1`. Replace this with the name of the editor from which the user is inserting the image. See Also: “Appendix A: Naming the eWebEditPro+XML Editor” on page 778.

```
<script language="JavaScript1.2">
<!--
function insertfile()
{
    top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtpath.value,false,"","IMAGE"
,0,0); window.close();
}
-->
</script>
```

Be sure to specify the parameters for `insertMediaFile`.

| Parameter | Value in this example |
|--------------------|----------------------------|
| file location | <code>txtpath.value</code> |
| is the file local? | <code>false</code> |
| file title | <code>" "</code> |
| file type | <code>" IMAGE "</code> |
| width | <code>0</code> |
| height | <code>0</code> |

NOTE

By entering zero (0) as the image’s width and height, the administrator is allowing the image to retain its original dimensions. The user can edit these values in the Picture Properties dialog box, which appears when the image is inserted.

(For more information, see “Specifying an Image to Insert” on page 491.)

- Enter text to prompt the user to specify the path to the image. For example

Enter path to image file:

- Create an input field to accept the user’s input. For example

```
<input type=text name="txtpath" size=30 value="">
```

- Create a button to invoke the `insertfile` function.

```
<input type=button name="btninsert" value="insert" onclick="insertfile()">
```

- Open the config.xml file. Within the `mediafiles` feature, at the `transport type` attribute, enter the path to the .htm file relative to local host. Place quotes around the path. For example

```
<transport type="/ewebeditpro5/imageselection.htm">
```

As a result, the following screen appears when the user presses the **Select New File** button on the Picture Properties dialog box.



WARNING!

If, while identifying an image, the user enters a pathway in a field used by JavaScript, the user *must* enter two backslash characters wherever they would normally enter one. As an alternative, the JavaScript could convert the backslash characters.

When the user enters a path to an image and clicks the **insert** button, the `insertMediaFile` command passes the image file information to the Picture Properties dialog box.

Below is the. htm file that you would use to implement this version of image selection.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>Untitled</title>
  <script language="JavaScript1.2" src="ewebeditpro.js">
  </script>
  <script language="JavaScript1.2">
  <!--
  function insertfile()
  {

top.opener.eWebEditPro.instances[ "MyContent1" ].insertMediaFile(txtpath.value,false,"",
"IMAGE",0,0);
    window.close();
}
-->
</script>
</head>

<body>
enter in a path:
<input type=text name="txtpath" size=30 value="">
<input type=button name="btninsert" value="insert" onclick="insertfile()">
</body>
</html>
```

For reference, the following illustrates the `mediafiles` section of the configuration data.

```

<mediafiles>
  <command name="cmdmfumedia" style="icon" visible="true">
    <image key="picture"/>
    <caption localeRef="btnTxtrunapp">Image File</caption>
    <toolTipText localeRef="btnrunapp">Image File</toolTipText>
  </command>
  <!-- 0 is unlimited size -->
  <maxsizek>1000</maxsizek>
  <validext>gif,jpg,png,jpeg,jif</validext>
  <mediacconfig enabled="true" allowededit="true"/>
  <!-- If this section is not defined it will default to FTP with no settings -->
  <!-- The attribute 'type' values "ftp" and "file" are handled within the editor. -->
  <!-- The scripting will load the page specified in the type attribute. -->
  <transport enabled="true" type="/ewebeditpro5/imageselection.htm"
confirmation="true" xfer="binary" pasv="true">
    <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
    <!-- blank for user entry -->
    <username encrypted="true"></username>
    <password encrypted="true"></password>
    <!-- Set to 0 for default port number -->
    <port>0</port>
    <!-- Upload location is: [domain]+[xferdir]+[filename] -->
    <domain></domain> <!-- e.g., ftp.mydomain.com -->
    <!-- Directory transferred into relative to domain -->
    <xferdir src="[eWebEditProPath]/upload"/>
    <!-- Referencing a file through HTTP is: [webroot]+[filename] -->
    <!-- if webroot is blank then it defaults to xferdir value -->
    <webroot src=""/>
    <!-- Possible values for resolvepath are: full, host, local, given -->
    <resolvemethod value="local" src=""/>
  </transport>
</mediafiles>

```

Example 2: File Size Restriction, No Saving to Database

In this example, the user inserts an image from a remote directory. The Web master sets a maximum image size of 100 Kb. If the user tries to insert an image larger than 100 Kb, an error message appears and the insertion is terminated.

NOTE

You can also use the `mediafiles` feature of the configuration data to limit the file types that users can insert, using the `validext` attribute. You implement this restriction in the same way you implement maximum file size.

To incorporate this version of image selection, follow these steps.

1. Within the `ewebeditpro5` directory, create an .htm file, for example, `imageselect_100kb.htm`.

2. Within the document's head tags, include the `ewbeditpro.js` file.

```
<script language="JavaScript1.2" src="ewebeditpro.js">
</script>
```

3. Still within the document's head tags, create a JavaScript function (in this example, `sizeisok`) that checks the size of the file selected by the user. If it exceeds 100 Kb, return false; otherwise, return true.

Note that in the example below, the variable `maxsize` refers to the `maxsizek` attribute of the mediafile feature in the config.xml file. In Step 6, you set the value of the `maxsizek` attribute.

NOTE

In the following example, the editor name appears as `MyContent1`. Replace this with the name of the editor from which the user presses the Insert Picture button. See Also: "Appendix A: Naming the eWebEditPro+XML Editor" on page 778.

```
function sizeisok()
{
  var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();
  var maxsize = objmedia.getPropertyInteger("MaxFileSizeK");

  if ((objmedia.FileSize) > maxsize*1024)
  {
    return (false);
  }
  else
  {
    return (true);
  }
}
```

NOTE

If you are using Netscape, you cannot access the ActiveX objects (such as `objmedia.MaxFileSizeK` and `objmedia.FileSize`) directly. Instead, use one of the `getProperty` methods to retrieve these values. (See "Using Netscape to Access Image Properties" on page 488.)

4. Create a function (in this example, `insertlocalfile`) that checks the value of the `sizeisok` function.

If the `sizeisok` function returns false, an error message appears ("File is too large."). If the function returns true, the `insertMediaFile` command passes the image file information to the Picture Properties dialog box.

```
function insertlocalfile()
{
  var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();

  objmedia.IsLocal = true;
  objmedia.SrcFileNameName = txtpath.value;
  if(sizeisok() == false)
  {
    alert("File is too large.");
  }
  else
  {

top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtpath.value,true,"","IMAGE",
0,0);
  window.close();
}
```

```
}
```

For a description of the rest of the code in this example, see steps 3 through 6 in “[Example 1: No Restrictions, No Saving to a Database](#)” on page 459.

5. Open the config.xml file. Within the mediafiles feature, at the transport type attribute, enter the path to the .htm file relative to local host. Place quotes around the path. For example

```
<transport type="/ewebeditpro5/imageselect_100kb.htm">
```

6. While in the config.xml file, set the value of the maxsizek attribute to 100.

```
<mediafiles>
<maxsizek>100</maxsizek>
```

Below is the entire .htm file that you would use to implement this version of image selection.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!--
How to set up in XML:
<mediafiles>
    ...
    <maxsizek>100</maxsizek>
    <transport type="/eWebEditPro5/imageupload_100kb.htm">
    </transport>
</mediafiles>
-->
<html>
<head>
    <title>Untitled</title>
<script language="JavaScript1.2" src="eWebEditPro.js">
</script>
<script language="JavaScript1.2">
<!--
function insertlocalfile()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();

    objmedia.IsLocal = true;
    objmedia.SrcFileLocationName = txtpath.value;
    if(sizeisok() == false)
    {
        alert("File is too large.");
    }
    else
    {

        top.opener.eWebEditPro.instances["MyContent1"].insertMediaFile(txtpath.value,true,"","IMAGE",0,0);
        window.close();
    }
}
function sizeisok()
{
    var objmedia = top.opener.eWebEditPro.instances["MyContent1"].editor.MediaFile();
    var maxsize    = objmedia.MaxFileSizeK;

    if ((objmedia.FileSize/1024) > maxsize)
    {
        return (false);
    }
    else
    {
        return (true);
    }
}
-->
</script>
</head>

<body>
enter in a path:
<input type=text name="txtpath" size=30 value="">
<input type=button name="btninsert" value="insert" onclick="insertlocalfile()">
</body>
</html>
```

For reference, the following illustrates the mediafiles section of the configuration data.

```
<mediafiles>
  <command name="cmdmfumedia" style="icon" visible="true">
    <image key="picture"/>
    <caption localeRef="btnTxtrunapp">Image File</caption>
    <toolTipText localeRef="btnrunapp">Image File</toolTipText>
  </command>
  <!-- 0 is unlimited size -->
  <maxsizek>100</maxsizek>
  <validext>gif,jpg,png,jpeg,jif</validext>
  <mediaconfig enabled="true" allowededit="true">
    <!-- If this section is not defined it will default to FTP with no settings -->
    <!-- The attribute 'type' values "ftp" and "file" are handled within the editor. -->
    <!-- The scripting will load the page specified in the type attribute. -->
    <transport enabled="true" type="/ewebeditpro5/ imageselection.htm" confirmation="true"
      xfer="binary" pasv="true">
      <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
      <!-- blank for user entry -->
      <username encrypted="true"></username>
      <password encrypted="true"></password>
      <!-- Set to 0 for default port number -->
      <port>0</port>
      <!-- Upload location is: [domain]+[xferdir]+[filename] -->
      <domain></domain> <!-- e.g., ftp.mydomain.com -->
      <!-- Directory transferred into relative to domain -->
      <xferdir src="[eWebEditProPath]/upload"/>
      <!-- Referencing a file through HTTP is: [webroot]+[filename] -->
      <!-- if webroot is blank then it defaults to xferdir value -->
      <webroot src="" />
      <!-- Possible values for resolvepath are: full, host, local, given -->
      <resolvemethod value="local" src="" />
    </transport>
  </mediafiles>
```

Example 3: FTP

You can implement image selection using FTP. To do this, enter **FTP** at the `type` attribute of the `mediafiles` feature of the configuration data. Enter the additional FTP information, such as domain, user name, port, and upload location in the `mediafiles` section of the configuration data.

Implementing FTP image selection can vary widely, depending on your system. Therefore, the Web master should determine how best to implement FTP-based image selection.

The next section provides an example of how to set up the configuration data for a typical FTP site. This example assumes that the FTP site and the Web site share the same physical server.

See Also:

- “[FTP File Upload](#)” on page 472

Minimum Configuration Requirements for FTP

These are the minimum configuration requirements if you use the FTP upload mechanism.

- The FTP site and the file's Web reference site must share the same physical server.
- The FTP server must be configured to allow access to a location that is also accessible through a Web browsing mechanism, that is, HTTP. If the FTP server is set to start in a directory structure that cannot be reached by a Web browser, the uploaded images cannot be displayed.

Server Configuration

Assume that FTP is set up with these parameters.

Parameter	Value
Domain	ftp.mydomain.com
Physical FTP Root	c:\inetpub\www\ftp
Images reside in	/shared/images
Physical image location	c:\inetpub\www\ftp\shared\images
Connection Port	Standard FTP Port
Data Transfer Style	Binary Data
Connection Mode	Must use passive mode for firewall

Assume that the Web site is set up with these parameters.

Domain	www.mydomain.com
Physical WWW Root	c:\inetpub\www
Page Location (Base URL)	/public/pages
Physical Page Location	c:\inetpub\www\public\pages

To implement the above configuration, you would set these values in the `mediafiles` section of the configuration data.

```
<transport type="ftp" xfer="binary" pasv="true">
<domain>ftp.mydomain.com</domain>
<xferdir src="/shared/images"/>
<webroot src="http://www.mydomain.com/public/pages"/>
```

Notice that since this example uses the standard FTP port, it does not include the `port` element.

Restriction Settings

To continue with the example, the administrator wants to add these restrictions to any uploaded images.

File Extensions	gif, jpg
Maximum File Size	12K
Login	User must log in to FTP account
File Referencing	All reference paths are relative to the local page

To implement these restrictions, you would set these values in the `mediafiles` section of the configuration data.

```
<validext>gif,jpg</validext>
<maxsizek>12</maxsizek>
<username></username>
<password></password>
<resolvemethod value="local"/>
```

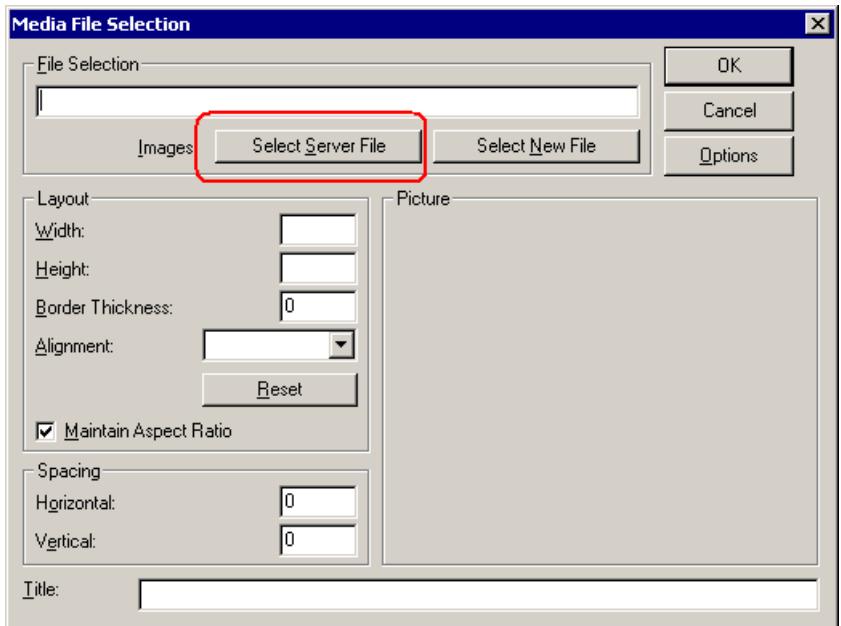
User Interface Control

The administrator does not want to let the user review any of the settings. The login dialog must be shown for the user to log in.

```
<mediaconfig enabled="true" allowededit="false"/>
```

Selecting Files from the Server

If you implement image selection using FTP, the Media File Selection dialog displays a **Select Server File** button that lets the user insert an image stored on the server.

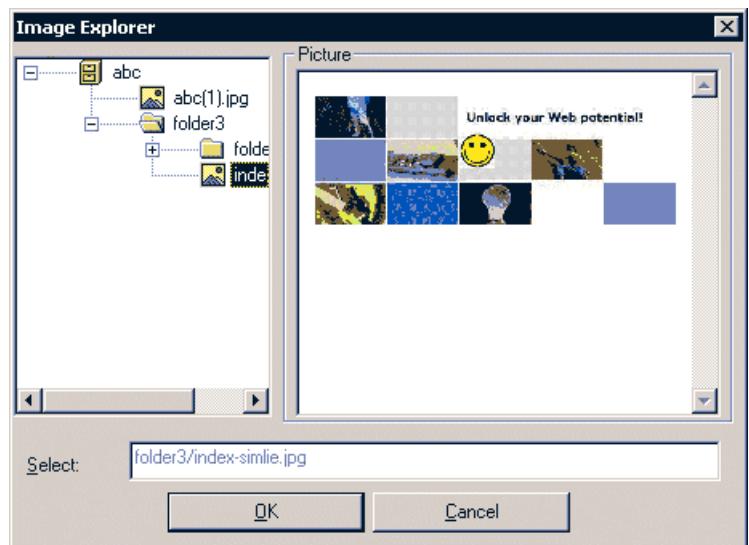


If a user clicks the button, a second screen displays the folder tree with the folders, sub-folders and their files in the FTP directory. The FTP folder is defined in the `xferdir` element. See Also: "["Xferdir Element"](#) on page 504.

NOTE

The default display name for the FTP Root folder is "Server." To modify it, use the `xferDispName` attribute of the `xferdir` element.

The user can select any image from the folder structure and preview it before inserting.



FTP Configuration in XML

The sample configuration described above uses this example MediaFiles section of the configuration data.

```
<mediafiles>
    <command name="cmdmfumedia" style="icon" visible="true">
        <image key="picture"/>
        <caption localeRef="btnTxtrunapp">Image File</caption>
        <toolTipText localeRef="bthrunapp">Image File</toolTipText>
    </command>
    <maxsizek>12</maxsizek>
    <validext>gif,jpg</validext>
    <mediaconfig enabled="true" allowededit="false"/>
    <transport type="ftp" xfer="binary" pasv="true">
        <username></username>
        <password></password>
        <port>0</port>
        <domain>ftp.mydomain.com</domain>
        <xferdir src="/shared/images"/>
        <webroot src="http://www.mydomain.com/public/pages"/>
        <resolvemethod value="local"/>
    </transport>
</mediafiles >
```

Example 4: Database Samples

When you install **eWebEditPro+XML**, you have an option to install database samples for your platform. For example, if you are running ASP, you can install ASP database samples.

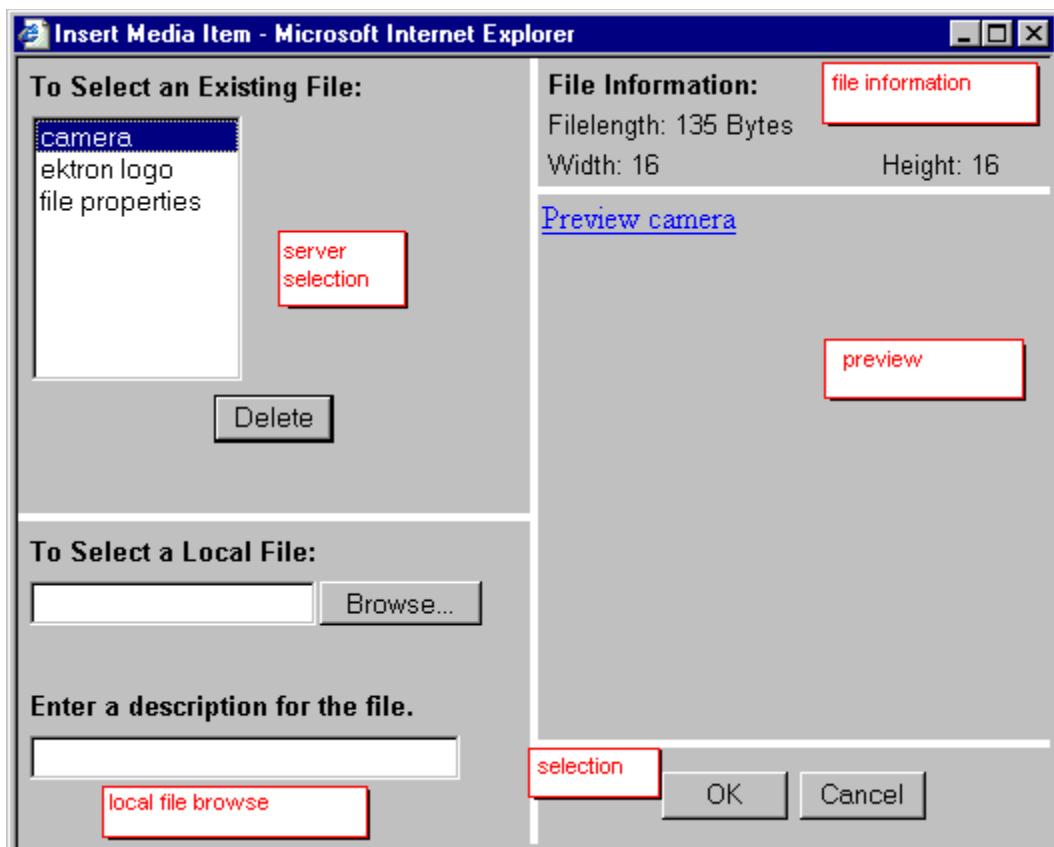
See Also: “[ASP](#)” on page 474

If you install database samples, a sample image selection screen is provided. (Where the image selection screen fits into the workflow of selecting an image is explained in Step 4 of “[How Image Selection Works](#)” on page 455.)

The sample screen lets the user select images from local directories or a server, upload files to a server, and preview an image before returning to the Picture Properties dialog box.

You can use the sample image selection screen as is, or modify it as needed for your users.

Below is the ASP sample image selection screen, with callout boxes to label the areas of the screen.



The following table describes the files that make up the ASP database sample. Samples for other platforms use essentially the same files -- only the file extensions are different.

Frame Name	File Name	Function - Allows the user to	Operation
server selection	medialist.asp	Select a file that resides on the server.	<p>Retrieves titles of all media files in the database, then builds an option list box for displaying the titles.</p> <p>The user can highlight the desired title. When the user highlights a title, the preview frame, the local file browse frame, the file information frame, and the selection frame are updated to reflect the selection.</p>

Frame Name	File Name	Function - Allows the user to	Operation
local file browse	mediauploader.asp	Choose a file from the local system. The local file is uploaded before it is inserted into the editor.	Lets the user choose a local file and assign it a title. When the user enters a local file, the server selection frame, the file information frame, the preview frame, and the selection frame are updated to reflect the selection.
file information	mediainformation.asp	View file information, including its length in bytes and, if the file is an image, its width and height in pixels.	Displays information about file the user selected, whether the file is server-based or local.
preview	mediapreview.asp	Preview the selected file.	Lets the user preview the highlighted file before selecting it.
selection	mediainsert.asp	Select a file. If a local file is selected, the file is uploaded before it is inserted into the editor.	Allows the user to select a server file or a local file. It also ensures that a title has been entered if the user selects a local file.

Implementing Image Upload

This section describes the following methods and options for enabling users to upload images and other files to your Web server.

- [FTP](#)
- [HTTP](#)
 - [ASP](#)
 - [ColdFusion](#)
 - [other Web servers](#)

Security issues surrounding each approach are explained.

FTP File Upload

You can use FTP (file transfer protocol) to copy files from the user's (or client) computer to the Web server. The Web server must have an FTP server to establish a connection and receive a file from the client computer. Many server operating systems provide an FTP server. Commercial FTP server software is also available.

eWebEditPro+XML can notify the server when a file is uploaded via FTP. This capability allows the server to update a database with the list of uploaded files.

To enable this notification, implement the `eWebEditProMediaNotification` function in JavaScript. This function opens a dynamic Web page and passes file information to the server, typically through URL parameters.

Security with FTP

Usually, you have an FTP account with a user name and password. When uploading files through **eWebEditPro+XML**, your FTP user name and password must be specified. To keep them secret, use Ektron's encryption program to scramble your user name and password.

Enter the user name and password in the `username` and `password` elements of the `mediafiles` feature of the configuration data. An example appears below.

```
<mediafiles>
  .
  .
  <transport enabled="true" type="ftp" confirmation="true" xfer="binary" pasv="true">
    <!-- Encrypt username and password using Ektron's encrypt.exe program. -->
    <!-- blank for user entry -->
    <username encrypted="true">zVQjUOPG</username>
    <password encrypted="true">uDekdcUF</password>
  .
  .

```

You may download Ektron's encryption program and view the [Encryption User's Guide](#) from Ektron's Web site.

HTTP File Upload

You can use HTTP (the same protocol that displays a Web page) to upload image files from a user's computer to a Web server. All Web servers support HTTP, but they usually require additional software to receive files from a client computer. Many Web application servers, such as ASP and ColdFusion, provide functions to write files to the Web server's file system.

NOTE

If a user deletes an image from the images list, the image is removed from the database but not from the physical directory on the server.

Overview

HTTP image upload with **eWebEditPro+XML** uses standard Web pages. You can write your own or use the samples provided with **eWebEditPro+XML**.

Typically, Web pages that upload images or other files include the elements shown below. Note that

- the `enctype` must be "**multipart/form-data**"
- you must specify an action page

- the input type of "file" displays a text box for the file name and a **Browse** button that lets the user select a file to upload

```
<form name="MyFormName" method="post" action="MyActionPage.xyz"
      enctype="multipart/form-data" OnSubmit="return MyValidateFormData()">
  ...
  <input type="file" name="MyUploadFile" size="20" maxlength="256" align="MIDDLE">
  ...
</form>
```

ASP

Microsoft Active Server Pages (ASP) include the ability to write text files to the file system, but do not have the native ability to write binary files. Since images (GIF and JPG) and other files (such as, audio, video, and Microsoft Office documents) are binary, an additional component is required.

Older versions of **eWebEditPro+XML** on ASP used a proprietary method to upload image files via HTTP. A server-side COM DLL, EkFileIO.DLL, was required to save the image file.

With **eWebEditPro+XML** 2.0 and higher, image upload uses standard multipart form data to upload the file. As a result, you can use any commercially-available file upload software for ASP. A popular file upload package for ASP is FileUp, available from SoftArtisans at <http://www.softartisans.com/softartisans/saf.html>.

Ektron still provides a server-side COM DLL, EktronFileIO.DLL (note the name change), for file upload support on a Windows NT 4 or Windows 2000 server.

EktronFileIO.dll

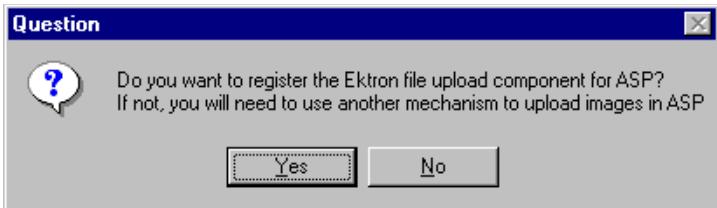
EktronFileIO.dll is a COM object that retrieves a file from multipart form data that has been submitted to the server. It then writes the file to the server's file system.

The COM object is created by the action page that is opened when the form is submitted. The ASP database sample, supplied with **eWebEditPro+XML** includes EktronFileIO.dll and an action page, medianotification.asp, to receive uploaded files. In it, you will see the object created using CreateObject("EktronFileIO.EkFile").

Registering EktronFileIO.dll

EktronFileIO.dll adds information to the Windows registry that allows the ASP page to create the COM object. As a result, you must register EktronFileIO.dll on the Web server before you can use it.

If you ran the Windows installation and responded Yes when the following dialog box appeared, the EktronFileIO.dll is already registered.



If you need to register EktronFileIO, open a command prompt and run regsvr32. By default, the EktronFileIO.dll is located in the `/ewebeditpro5/samples/asp/database` directory under the Web root, but it can reside anywhere on the server.

Here is the code you would enter to register EktronFileIO.dll if it is in the default directory.

```
cd \inetpub\wwwroot\ewebeditpro5\samples\asp\database
regsvr32 EktronFileIO.dll
```

Licensed owners of **eWebEditPro+XML** 2.0 and higher may download EktronFileIO.dll onto their Windows Web server.

Security with ASP

The image selection page with the Browse button should validate the file extension to upload. Security should also be in the ASP page that is the form's action page.

The ASP page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png (see ["Validext Element" on page 495](#)). You may also want to allow document files, such as, doc and pdf extensions, or media files, such as wav, ram, and asf.

You should not allow ASP or HTML files to be uploaded; a malicious person could gain control over the Web server and cause damage.

For best security, only allow authorized users to access a page with **eWebEditPro+XML** on it. Windows Server provides a user authentication capability.

Alternatively, you could use FTP, which is protected with a password. Image upload can be disabled altogether on **eWebEditPro+XML** if needed.

The ASP database sample supplied with **eWebEditPro+XML** 2.0 and higher includes an action page, medianotification.asp, to receive uploaded files.

The EktronFileIO's API

EktronFileIO.dll is a Visual Basic 6.0 utility that allows ASP (and other platforms) to write a multipart form file upload to the server's file system. The DLL handles the following tasks:

- Extracts one "uploaded file" from the supplied data stream
- Saves the extracted file to a user-designated directory

- Returns form field values. ASP cannot access a form field if BinaryRead is used anywhere in the page.
- Handles a name conflict
- Handles permission setting on the new file (not supported in this release)
- Handles error reporting

The API closely resembles the ColdFusion CFFILE function. The interface is as follows:

```
ReturnString = EkFileSave ("BinaryFormData", "FormFieldName", "DestinationDir", ErrorCode,
["NameConflict"], ["AcceptType"], ["FilePermissionSetting"],
["FileAttributes"])ReturnedFormFieldValue = fileObj.EkFormFieldValue("BinaryFormData",
"FormFieldname", ErrorCode)
```

Parameter	Data Type	Required / Optional	Description
BinaryFormData	Variant (String)	Required	The entire form data in binary form
FormFieldName	Variant (String)	Required	The name of the field used in the original form
DestinationDir	Variant (String)	Required	The fully qualified path (for example c:\inetpub\wwwroot\test)
ErrorCode	Variant (Number)	Required	A user-supplied variable. This is set to 0 (zero) for successful execution. Otherwise, it is set to one of the error codes listed below.
NameConflict	Variant (String)	Optional	Determines the behavior when a requested filename conflicts with an existing file.
AcceptType	Variant (String)	Optional	Determines which file types the upload will accept (for example, image/gif, application/msword). Not supported in this release.
FilePermissionSetting	Variant (String)	Optional	Not supported in this release.

Parameter	Data Type	Required / Optional	Description
FileAttributes	Variant (String)	Optional	Not supported in this release.
ReturnString	Variant (String)	Always returned	If ErrorCode (see above) is 0 (zero), this contains the filename that stores the file, including the full path. If ErrorCode is not zero, this contains a matching error string.

Error Codes

ErrorCode	Description	Internal/External COM object Error
101	"Error: Form Field Name not found." The user-requested form field cannot be located.	Internal COM object error
102	"Error: Cannot locate 'Content-Disposition' text." The HTTP "Content-Disposition" header cannot be located in the form.	Internal COM object error
103	"Error: Cannot locate filename in form field." The user-requested form field does not contain an associated filename. The requested form field may not be type "File".	Internal COM object error
104	"Error: Bad Form Filename." The filename in the user-requested form field is not properly formatted.	Internal COM object error
105	"Error: Cannot locate binary file data." An error was encountered while searching for the associated binary file data.	Internal COM object error
106	"Error: File Already Exists" The filename that the form requested is already in use, and the COM object is not allowed to rename the file.	Internal COM object error
XXX	Windows system errors reported while writing or deleting the requested filename.	External: Code and error string returned by the operating system.

Using EktronFileIO for Your Own Image Uploads

Often, the uploading of files, such as images, is made possible by a set of Web pages created for a site. This section describes how to use the EktronFileIO DLL (installed with the **eWebEditPro+XML** server-side installation) in its most basic sense. The **eWebEditPro+XML** editor is not part of these samples. Gaining familiarity with the DLL helps you to integrate it into your own external upload mechanism.

This section explains:

- Creating an ASP page that asks the user to select a file
- Creating an ASP page that performs the upload
- Examining the EktronFileIO upload method
- Examining return values for errors and file name changes
- Retrieving the values of field items on a submitted form

When using EktronFileIO, keep these in mind:

- The ASP mechanism processes the posted information
- The EktronFileIO must be registered on the server (see ["Registering EktronFileIO.dll" on page 474](#))
- For security reasons, files cannot be uploaded without user intervention
- All data sent to EktronFileIO is contained in a form that is posted to the server

Using EktronFileIO involves four steps:

1. [Creating a Selection Web Page](#)
2. [Creating a Form with a File Selection Field Item](#)
3. [Creating an ASP Page to Activate the Posted Upload](#)
4. [Providing Upload Feedback](#)

Step 1: Create a Selection Web Page

In this step, we create a simple ASP page that contains an upload and cancel button. The Upload button is a submit button.

Below is the HTML for this page. Save the file as simpleupload.asp within the server's Web directory.

```
<html>
<head>
<title>EktronFileIO Upload Example</title>
</head>
<body>
<h1>Upload a File with EktronFileIO</h1>
<br>
<input type="submit" name="btnupload" value="-- Upload --">
<input type="submit" name="btncancel" value="Cancel">
</body>
</html>
```

Step 2: Create a Form with a File Selection Field Item

Within the HTML that the user interacts with, we need to create a form that is submitted to the server. The form contains the name of the file to upload and, optionally, other information we may want to use.

The form must contain these attributes and values:

```
method="POST"
enctype="multipart/form-data"
```

The following attributes are also required, but their values depend on the implementation. For our example, they contain these values:

```
action="performupload.asp"
name="frmupload"
```

The ‘action’ attribute value, “performupload.asp”, specifies the page that activates the upload mechanism. (We will create this page in Step 3.) We are naming the form “frmupload”.

Here is the form added to the HTML:

```
<html>
<head>
    <title>EktronFileIO Upload Example</title>
</head>
<body>
<h1>Upload a File with EktronFileIO</h1>
<form action="performupload.asp" method="POST" enctype="multipart/form-data" name="frmupload">
<br>
    <input type="submit" name="btnupload" value="-- Upload --">
    <input type="submit" name="btncancel" value="Cancel">
</form>
</body>
</html>
```

The only required form item (other than the submit button) is a FILE input item. The EktronFileIO uses this input type to retrieve the name of the file to upload. Since this input item requires the user to physically select a file, it prevents files from being uploaded erroneously from the client.

Below, the FILE input item is highlighted in red.

```
<html>
<head>
    <title>EktronFileIO Upload Example</title>
</head>
<body>
<h1>Upload a File with EktronFileIO</h1>
<form action="performupload.asp" method="POST" enctype="multipart/form-data" name="frmupload">

    Please select a file:<br>
    <!-- This is the only required field.
        It contains the selected file to upload. -->
    <input type="File" name="uploadfilephoto" size="20" maxlength="256">

    <br>
    <input type="submit" name="btnupload" value="-- Upload --">
    <input type="submit" name="btncancel" value="Cancel">

</form>
</body>
</html>
```

```
</form>
</body>
</html>
```

The new lines also give the user feedback about what to do.

The FILE input item contains the name and location of the local file to upload. The EktronFileIO DLL reads this information from the form submittal and uploads from the source location. For this sample, we hard code the destination location.

This is the full page for asking the user what file to upload. Next, we create the ASP page that is called when a post event activates the upload mechanism.

Step 3: Creating an ASP Page to Activate the Posted Upload

The ASP page created above calls for a second page to be loaded when the form we defined is posted. The second page contains ASP code that interacts with the EktronFileIO DLL object and displays any feedback we need.

To create the second page, begin by creating a basic ASP page. Save this page as `performupload.asp` in the same directory as the page created above. (This is the same name that we placed in the action attribute of the form element above.)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Page That Activates Upload</title>
</head>
<body bgcolor="Silver">
    <h1>Uploaded File</h1>
</body>
</html>
```

At this point, test the page to ensure that it loads when a post occurs. (If this mechanism does not work, it does not matter how much ASP you place into your pages.) To test it, load the first page in a browser. When you press the "Upload" button, the second page with a gray background and the "Uploaded File" header line should load.

Now, we'll add the ASP that activates the upload with the EktronFileIO DLL.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Page That Activates Upload</title>
</head>
<body bgcolor="Silver">
<h1>Uploaded File</h1>

<%
    Dim BinaryFormData, fileObj, ServerLocation
    Dim strReturnString, ErrorCode

    BinaryFormData = Request.BinaryRead(Request.TotalBytes)
    set fileObj = CreateObject("EktronFileIO.EkFile")
    ServerLocation = "/images"      ' Hard coded the location for this sample.

    strReturnString = fileObj.EkFileSave(BinaryFormData, "uploadfilephoto", _
    Server.MapPath(ServerLocation), ErrorCode, "makeunique")
%>

</body>
</html>

```

Here are some things to notice in the code:

- We hard coded the destination location. (See ["Making the Destination Location Dynamic" on page 483](#) to learn how to dynamically set this value.)
- We performed a binary read to load the file into the form.
- An object reference to EktronFileIO is created for the upload.
- The "uploadfilephoto" input item specifies which item has the file selection.
- The upload may change the file name, so the actual name returned is placed into the strReturnString variable.

You can upload any file to the "/images" location. Test this by uploading a file.

You can stop here if you like. This example continues to explain how to handle errors and give other feedback.

Step 4: Providing Upload Feedback

Error Handling

Errors are returned in the variable given to the EkFileSave method. In our example, it is the ErrorCode variable. Here, we use it to display a status.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Page That Activates Upload</title>
</head>
<body bgcolor="Silver">
<h1>Uploaded File</h1>
<%
    Dim BinaryFormData, fileObj, ServerLocation

```

```

        Dim strReturnString, ErrorCode

        Dim BinaryFormData, fileObj, ServerLocation
        Dim strReturnString, ErrorCode

        BinaryFormData = Request.BinaryRead(Request.TotalBytes)
        set fileObj = CreateObject("EktronFileIO.EkFile")
        ServerLocation = "/images"      ' Hard coded the location for this sample.

        strReturnString = fileObj.EkFileSave(BinaryFormData, "uploadfilephoto", _
        Server.MapPath(ServerLocation), ErrorCode, "makeunique")
    %>

    % if (0 = ErrorCode) then %>
        <h3>Load Succeeded</h3>
    <% else %>
        <h3><font color="Red">Load Failed with Error = <%= (ErrorCode)%></font></h3>
        <h3><font color="Red">Error Description = <%= (strReturnString)%></font></h3>
    <% end if %>
</body>
</html>

```

If there is an error, the returned string is an English description of it.

Displaying Selection Information from Field Items

The name of the file is retrieved using the EkFormFieldValue method. This returns the value of any field on the form.

Here is an example of using the method to display the file selected for upload.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Page That Activates Upload</title>
</head>
<body bgcolor="Silver">
<h1>Uploaded File</h1>
<%
    Dim BinaryFormData, fileObj, ServerLocation
    Dim strReturnString, ErrorCode, UploadFileName

    BinaryFormData = Request.BinaryRead(Request.TotalBytes)
    set fileObj = CreateObject("EktronFileIO.EkFile")
    ServerLocation = "/images"      ' Hard code the location for this sample.

    UploadFileName = fileObj.EkFormFieldValue(BinaryFormData, "uploadfilephoto", ErrorCode)
    response.write("Uploading the file: " & UploadFileName)

    strReturnString = fileObj.EkFileSave(BinaryFormData, "uploadfilephoto", _
    Server.MapPath(ServerLocation), ErrorCode, "makeunique")
%>
<% if (0 = ErrorCode) then %>
    <h3>Load Succeeded</h3>
<% else %>
    <h3><font color="Red">Load Failed with Error = <%= (ErrorCode)%></font></h3>
<% end if %>

```

```
</body>
</html>
```

Displaying the Resulting File Name

When a file is loaded, the "makeunique" option modifies the file name to be unique if it exists. The string returned from the EkFormFieldValue call contains the name of the file as it exists on the server. This does not include the path.

This name, with the destination path, should be used for any reference values from HTML.

```
<% if (0 = ErrorCode) then %>
    <h3>Load Succeeded</h3>
    <p>The file now exists at: <%= (ServerLocation)%>/<%= (strReturnString)%></p>
<% else %>
    <h3><font color="Red">Load Failed with Error = <%= (ErrorCode)%></font></h3>
    <h3><font color="Red">Error Description = <%= (strReturnString)%></font></h3>
<% end if %>
```

Making the Destination Location Dynamic

We now use what we know to make the destination directory not hard coded. First, edit the first ASP file, simpleupload.asp, that we created. Add to the form an input field that contains the path.

In this example, we'll use a "hidden" field without a path value. (This could be a text field if we wanted user intervention.)

```
<html>
<head>
    <title>EktronFileIO Upload Example</title>
</head>
<body>
    <h1>Upload a File with EktronFileIO</h1>
    <form action="performupload.asp" method="POST" enctype="multipart/form-data" name="frmupload">
        Please select a file:<br>
        <!-- This is the only required field.
        It contains the selected file to upload. -->
        <input type="File" name="uploadfilephoto" size="20" maxlength="256">

        <input type="hidden" name="dest_loc" value="/images">

        <br>
        <input type="submit" name="btnupload" value="-- Upload --">
        <input type="submit" name="btncancel" value="Cancel">

    </form>
</body>
</html>
```

In the ASP examples, the destination field is loaded with the content of the Media File 'webroot' attribute value.

After saving these changes, edit the ASP file. Next, edit the ASP file that activates the upload, performupload.asp. Change the ServerLocation variable from a hard coded value to the value of the dest_loc field in the submitted form.

```
<% response.buffer = false %>
```

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Page That Activates Upload</title>
</head>
<body bgcolor="Silver">
<h1>Uploaded File</h1>
<%
    Dim BinaryFormData, fileObj, ServerLocation
    Dim strReturnString, ErrorCode, UploadFileName

    BinaryFormData = Request.BinaryRead(Request.TotalBytes)
    set fileObj = CreateObject("EktronFileIO.EkFile")

    ServerLocation = fileObj.EkFormFieldValue(BinaryFormData, "dest_loc", ErrorCode)

    UploadFileName = fileObj.EkFormFieldValue(BinaryFormData, "uploadfilephoto", ErrorCode)
    response.write("Uploading the file: " & UploadFileName)

    strReturnString = fileObj.EkFileSave(BinaryFormData, "uploadfilephoto", _
        Server.MapPath(ServerLocation), ErrorCode, "makeunique")
%>

<% if (0 = ErrorCode) then %>
    <h3>Load Succeeded</h3>
    <p>The file now exists at: <%=ServerLocation%>/<%=strReturnString%></p>
<% else %>
    <h3><font color="Red">Load Failed with Error = <%=ErrorCode%></font></h3>
    <h3><font color="Red">Error Description = <%=strReturnString%></font></h3>
<% end if %>
</body>
</html>

```

Now, the ASP page that activates the upload dynamically retrieves the destination from the dest_loc field.

NOTE

Regarding the destination location given for the upload: the path specified *must* be visible to IIS, either physically or virtually. If it is not, there is no access for uploads. This is why a path like "http://localhost" does not work.

Conclusion

From here the site must implement other options, such as file type checking, database updating, and any other required functionality.

ColdFusion

Macromedia/Allaire ColdFusion server has a CFFILE feature that enables you to save files to the server's file system. See the ColdFusion server documentation for details on CFFILE.

The ColdFusion database sample supplied with **eWebEditPro+XML** includes an action page (medianotification.cfm) and a custom tag file (ewebeditprouploadfile.cfm) to receive uploaded files. In it, you see the <cffile action="UPLOAD" . . .> tag.

Security with ColdFusion

The image selection page with the Browse button should validate the file extension to be uploaded. Security should also be in the ColdFusion page that is the form's action page. The ColdFusion page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png. You may also want to allow document files, such as, doc and pdf extensions, or media files, like, wav, ram, and asf (see “[Validext Element](#)” on page 495).

You should not allow CFM or HTML files to be uploaded; a malicious person could gain control over the Web server and cause damage.

For best security, you should only allow authorized users to access a page with **eWebEditPro+XML** on it. Most Web servers provide user authentication.

Alternately, you could use FTP, which is protected with a password. If needed, you can disable Image Upload. The ColdFusion administrator can enable or disable the CFFILE tag.

Other Web Servers

NOTE Ektron provides a PHP and JSP image upload sample files. See the Developer's Page on Ektron's Web site for details (http://www.ektron.com/index.cfm?doc_id=654).

Your Web application server must support file upload and provide an ability to write binary files to the server's file system. Files are uploaded using HTTP in a Web page form using multipart form data. Check your documentation for instructions. Third party software may also be available.

Security

The image selection page with the Browse button should validate the file extension to upload. Security should also be in the dynamic Web page that is the form's action page.

The page should check the file type and only accept files that are safe, such as image files with extensions: gif, jpg, or png. You may also want to allow document files, such as, doc and pdf extensions, or media files, like, wav, ram, and asf (see “[Validext Element](#)” on page 495).

You should not allow dynamic pages and HTML files to be uploaded; a malicious person could gain control over the Web server and cause damage.

For best security, only allow authorized users to access a page with **eWebEditPro+XML** on it. Most Web servers provide user authentication.

Alternatively, you could use FTP, which is protected with a password. Image upload can be disabled altogether on **eWebEditPro+XML**, if needed.

Manipulating Media File Methods and Properties

The Media File Object methods and properties contain information about the file, the source location, and the destination.

The object automatically parses the path and uses the values from some of the properties to determine a transfer destination path and a reference path. Initial values for several of these parameters are specified in the mediafiles feature of the configuration data.

For more information, see "[Media File Object](#)" on page 20.

Using Local or Given Image Path Resolutions

To learn when to use the *local* or *given* image path resolution type, it is important to understand that image paths are resolved in one of three ways.

- full path
- relative to the host
- relative to the local page location

Below is an example, based on these file locations, of a page whose image path is resolved in each way.

Page Location: `http://www.yourcompany.com/pages/ewebeditpro5`

Image Location: `http://www.yourcompany.com/images/gifs`

Image File Name: `happy.gif`

Resolution Type	Image Path in HTML
Full	<code>http://www.yourcompany.com/images/gifs/happy.gif</code>
Host	<code>/images/gifs/happy.gif</code>
Local	<code>.../..../images/gifs/happy.gif</code>

Your choice of a resolution type is determined by the needs of the site and the publishing process. Use the `ResolveMethod` property to define a resolution of the image path. See Also: "[Method: resolvePath](#)" on page 116

Base URL

Another concept to understand is the *Base URL*, the location where a page is being edited.

In the example above, the base URL is `http://www.yourcompany.com/pages/ewebeditpro5`. To get from the Base URL location to the image location relatively, use this syntax: ".../.../images/gifs".

Given Resolution Type

The given resolution type is an abstract version of the local type. It produces a relative path to images from a directory other than the Base URL.

The given type uses the attribute, `src`, whose value is the path to the intended publishing location. The `src` attribute replaces the Base URL.

When using the given type, set all paths relative to the specified location rather than the Base URL.

The given type does not change the images' reference location or upload location.

Below is the above example, based on these file locations again, this time using the given resolution type.

Page Location: `http://www.yourcompany.com/pages/ewebeditpro5`

Image Location: `http://www.yourcompany.com/images/gifs`

Image File Name: `happy.gif`

Given `src` Location: `/publish/articles/local/sports`

Resolution Type	Image Path in HTML
Full	<code>http://www.yourcompany.com/images/gifs/happy.gif</code>
Host	<code>/images/gifs/happy.gif</code>
Given	<code>.../.../.../images/gifs/happy.gif</code>

Since the `src` path may be at a different level or location than the editing location, all paths stored in the HTML are relative to the given location rather than the editing location.

As you can see, if you use the given resolution type, the paths in the HTML may not match the actual paths to the files. If so, the images do not appear in the editor but do appear when the page is published to the destination location.

Conditions for Using Given Resolution Type

To use the given resolution type, all of the following conditions must be true.

- The images do not move when the HTML source page moves.

- The HTML source page is published in a different directory level from the directory in which it is edited.
- It is acceptable to have images not appear when a page is being edited.
- You know where the HTML source page is published.

If any condition is not true, you should *not* use the given resolution type. Instead, use the local or full resolution type.

Programmatically Accessing Media File Properties

The Media File Object provides access to image properties relating to the file and the upload process. Values set for these properties affect the operation of the editor.

["Media File Object" on page 20](#) lists the properties. You can set default values for most properties in the configuration data.

This section provides the following topics, which explain how to *programmatically* access the image properties under various circumstances.

- [Accessing the Media File Object](#)
- [Using Netscape to Access Image Properties](#)
- [The Entry Point for Using External Scripts](#)
- [Setting External Page Parameters](#)
- [Changing the Transfer Method on the Fly](#)
- [Specifying an Image to Insert](#)
- [Modifying the Upload Directory](#)

Accessing the Media File Object

You gain access to the media file object properties programmatically via the `MediaFile` method in the `eWebEditPro+XML` control.

```
Function getValidExtensions(seditorname)
{
  var objMedia = top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile();
  return(objMedia.getPropertyString("ValidExtensions"));
}
```

See Also: ["Appendix A: Naming the eWebEditPro+XML Editor" on page 778](#); ["Media File Object" on page 20](#)

Using Netscape to Access Image Properties

Within Netscape, the Esker ActiveX plug-in converts the ActiveX control to a plug-in that Netscape can interpret. As a result, you cannot access Media File Object image properties directly. Instead, use the `getProperty` and `setProperty` methods listed below.

```
setProperty(strName, strValue)
```

```

getProperty(strName) as Object
getPropertyInteger(strName) as Integer
getPropertyString(strName) as String
getPropertyBoolean(strName) as Boolean

```

Below are examples of their usage.

```

bIn =
eWebEditPro.instances[sEditorName].editor.MediaFile().getPropertyBoolean("HandledInternally");
bUpload = objMedia.getPropertyBoolean("AllowUpload");
sExt = objMedia.getPropertyString("ValidExtensions")
iSz =
top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile().getPropertyInteger("MaxFileSizeK");
objMedia.setProperty("SrcFileLocationName", cStr);
eWebEditPro[sEditor].MediaFile().setProperty("TransferMethod", "mediamanager.cfm");

```

Similar property access is done within Java applications. The Java Bean file provides the functionality for accessing properties.

See Also: ["Method: getProperty" on page 90](#); ["Method: setProperty" on page 124](#)

Entry Point for Using External Scripts

The ewebeditpromedia.js file contains the entry point for external scripting of image selection and upload. Its contents are below.

```

// Copyright 2000-2001, Ektron, Inc.
// Revision Date: 2001-04-03

// Media Upload Functionality
// Modify this file to customize file upload capability.

function eWebEditProMediaSelection(sEditorName)
{
    // The transfer method specifies what to load for the transfer.
    var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
    var XferMethod = objMedia.getPropertyString("TransferMethod");
    var sPageLoad = escape(XferMethod) + '?editorname=' + escape(sEditorName) +
'&upload=' + escape(objMedia.getPropertyBoolean("AllowUpload"));

    if(XferMethod != "")
    {
        window.open(sPageLoad, 'Images', "scrollbars,resizable,width=640,height=480");
    }
    else
    {
        alert('The Transfer Method value is empty. Please specify either "FTP" or a site
address that will handle the file selection.');
    }
}

```

The page value is specified in XML like this.

```

<features>
. . .
<mediafiles>

```

```

    . . .
        <transport type="samples/asp/database/mediamanager.asp">
    . . .
        </transport>
    </mediafiles>
</features>
```

You can also specify the transport type by modifying the TransferMethod property of the Media File Object. The ASP and ColdFusion samples demonstrate this.

See Also: ["Property: TransferMethod" on page 157](#)

Setting External Page Parameters

External pages can pass two parameters to help process the image request.

- Editor's Name (editorname)
- Upload Access (upload)

The parameters are passed like this.

```
mediamanager.cfm?editorname=MyContent1&upload=true
```

Use the (editorname) parameter to access the editor in scripts. The parameter is the name of the editor that processed the command to bring up the page. The name can be anything. In the sample files provided by Ektron, the name is MyContent1 or MyContent2.

See Also: ["Appendix A: Naming the eWebEditPro+XML Editor" on page 778](#)

The Upload Access (upload) parameter specifies whether the user can upload image files to the server.

See Also: ["Property: allowupload" on page 150](#)

Advanced users can specify their own parameters in the configuration data or set them in the XferType property of the Media File Object. Custom parameters *must* appear at the beginning of the parameter list. The two standard parameters are appended to the end of the list.

For example, a user wants to pass the domain name as a parameter. Here is how you would define this in the configuration data.

```
<transport type="mymediaupload.cfm?domain=mydomain">
```

Here is how you would define this in the script.

```
var objMedia = top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile();
objMedia.XferType = "mymediaupload.cfm?domain=mydomain";
```

Changing the Transfer Method on the Fly

This example shows how to specify a page while running the script.

```
function initTransferMethod(sEditor)
{
    eWebEditPro[sEditor].MediaFile().setProperty("TransferMethod", "mediamanager.cfm");
}
```

Programmatically Changing from the Default of FTP to the ASP Library

This sample is taken from the edit.asp page. It also exists in the editdesign.asp and edittemplate.asp. It uses the editor's interface (fascias) to modify what was in the configuration.

```
eWebEditPro.addEventHandler("onready", "initTransferMethod(eWebEditPro.event.srcName,
'mediamanager.asp', 'autoupload.asp')");
function initTransferMethod(sEditor, strURL, strAutoURL)
{
    if (eWebEditPro.instances[sEditor] != null)
    {
        // The GUI Selection method:
        eWebEditPro[sEditor].MediaFile().setProperty("TransferMethod", strURL +
"?autonav=" + escape(AutoNav) + "&defualtFolderId=" + defaultFolderId);

        // The Automatic Accept method:
        eWebEditPro.instances[sEditor].editor.MediaFile().AutomaticUpload().setProperty(
            "TransferMethod", strAutoURL);
        eWebEditPro.instances[sEditor].editor.MediaFile().AutomaticUpload().SetFieldValue(
            "folder_id", defaultFolderId);
    }
}
```

Specifying an Image to Insert

This JavaScript example shows how to insert an image that was loaded by an external mechanism.

```
Function useSelectedFile(seditorname, sfilename, stitle)
{
    //This will bring up the properties dialog and have the user confirm the insert.
    top.opener.eWebEditPro.instances[seditorname].insertMediaFile(sfilename, 0, stitle,
filetype[iloop], 0, 0);
}
```

The insertMediaFile function is defined in the core JavaScript. (See “[Method: insertMediaFile](#)” on page 94.)

Below is the code in the Core JavaScript.

The script must inform the Media File Object that the file about to be specified is remote. To do this, set the `IsLocal` property to **false**.

```

function eWebEditProEditor_insertMediaFile(strSrcFileLocation, bLocalFile,
strFileType, strFileName,
nWidth, nHeight)
{
    setTimeout('eWebEditPro.instances["' + this.name + '"].insertMediaFileDeferred("' +
strSrcFileLocation + '", "' + bLocalFile + '", "' + strFileName + '", "' + strFileType +
'", ' + nWidth + ', ' + nHeight + ')', 1);
}

function eWebEditProEditor_insertMediaFileDeferred(strSrcFileLocation,
        bLocalFile, strFileName, strFileType, nWidth, nHeight)
{
    // Place the file information into the media file object.
    // This is used for the insertion of the HTML.
    var objMedia = this.editor.MediaFile();

    objMedia.setProperty("IsLocal", bLocalFile);
    objMedia.setProperty("SrcFileName", strSrcFileLocation);
    objMedia.setProperty("FileName", strFileName);
    objMedia.setProperty("FileType", strFileType);
    objMedia.setProperty("ImageWidth", nWidth);
    objMedia.setProperty("ImageHeight", nHeight);

    this.editor.ExecCommand("cmdmfuinsert", strSrcFileLocation, bLocalFile);
}

```

This example also does not specify a width and height. If they are not specified, the properties dialog box offers to the user the ability to retrieve the file and determine the dimensions.

Modifying the Upload Directory

Here is an example of changing the upload and reference directory while executing a script.

For server-side functionality, such as ASP, JSP, ColdFusion, and PHP, the transfer directory and the reference directory should be set the same. Other upload functionality, such as FTP, may have these as two different directories. This sample assumes server-side functionality such as ASP or ColdFusion.

```

function SetTransferDirectory(seditorname, spathname)
{
    // This sets the transfer directory for the named editor.

    top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile().setProperty("Transfer
Root", spathname);

    // Since the upload and Web reference are the same, we should also
    // ensure that the reference path is the same.
    top.opener.eWebEditPro.instances[sEditorName].editor.MediaFile().setProperty("WebRoot",
spathname);
}

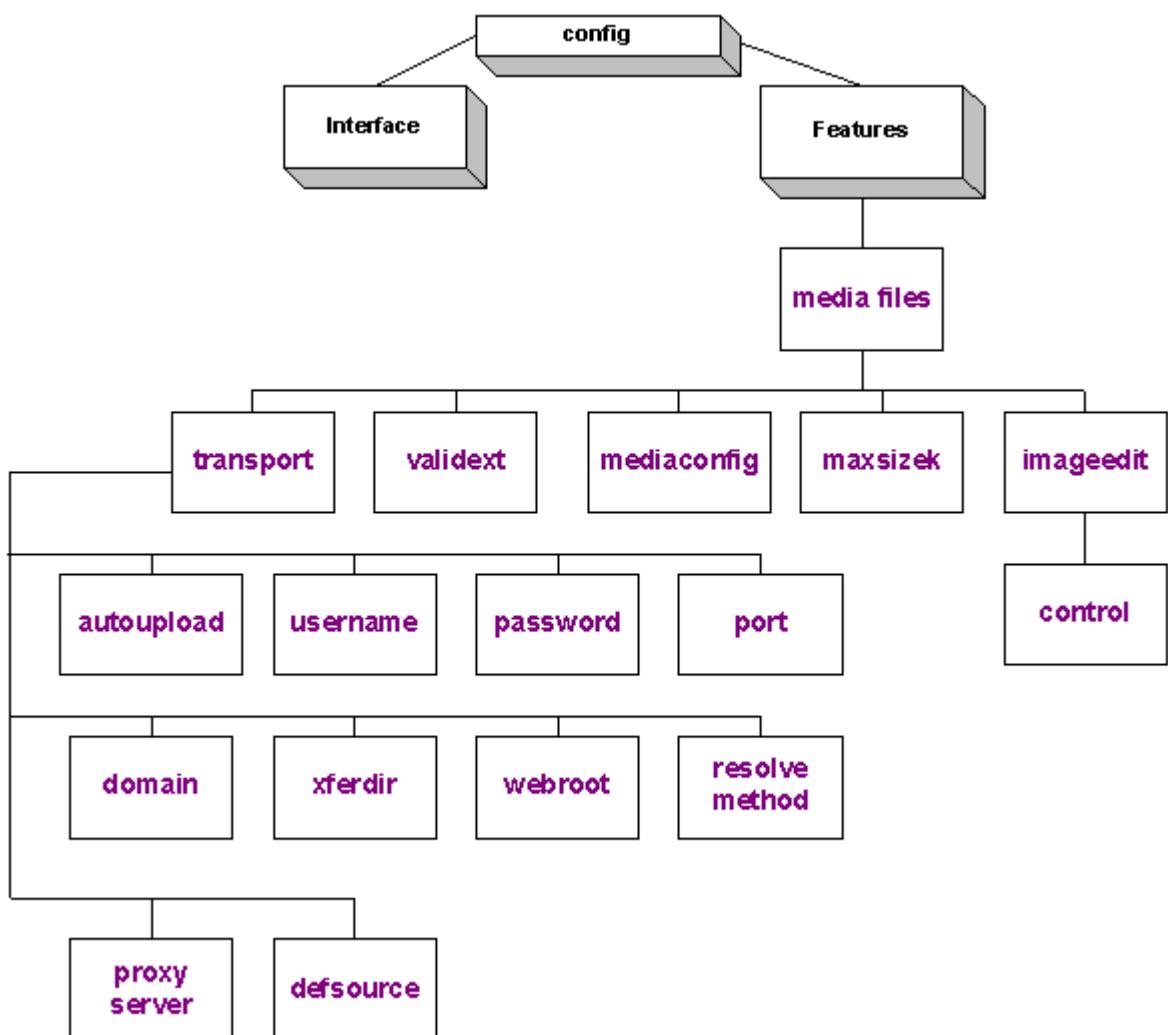
```

See Also: [“Appendix A: Naming the eWebEditPro+XML Editor” on page 778](#)

The Mediafiles Feature

This section describes the elements of the `mediafiles` feature of the configuration data. For an overview of the media files feature, see "Managing Images" on page 455.

Mediafiles Element Hierarchy



User Interface Elements in Alphabetical Order

Element	Description	For more information, see
autoupload	Defines operation of automatic image upload	"Autoupload Element" on page 498
control	Sets location of WebImageFX's configuration data file	"Control Element" on page 508
defsource	The default folder that appears when a user browses for a file on a local system	"Defsource Element" on page 506
domain	The domain name for the connection	"Domain Element" on page 503
imageedit	Defines WebImageFX	"Imageedit element" on page 508
maxsizek	Specifies the maximum file size allowed for upload	"Maxsizek Element" on page 496
mediaconfig	Controls the configuration dialogs	"Mediaconfig Element" on page 496
mediafiles	Defines the configuration options for the mediafiles feature	"Mediafiles Element" on page 495
password	Provides password for gaining access to server	"Password Element" on page 502
port	Specifies port to use for file transfers	"Port Element" on page 506
proxyserver	Specifies the proxy server to use.	"Proxyserver Element" on page 503
resolvemethod	Defines how to resolve file paths	"Resolvemethod Element" on page 507
transport	Defines mechanism for selecting and uploading media files	"Transport Element" on page 497
username	Provides user name for gaining access to server	"Username Element" on page 502

Element	Description	For more information, see
validext	Specifies valid extensions allowed for upload	"Validext Element" on page 495
webroot	Specifies path to use when referencing uploaded file	"Webroot Element" on page 505
xferdir	The destination directory on the server for the upload	"Xferdir Element" on page 504

Mediafiles Element

Description

Defines the configuration options for the `mediafiles` feature.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
```

Child Elements

`maxsizek`, `validext`, `mediaconfig`, `transport`, `imageedit`

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Defines whether the feature is enabled. If set to false , the feature is not available to the user.

Validext Element

Description

Specifies the valid extensions allowed for upload.

The editor removes wildcard characters and spaces from the list. So, for example, gif, .gif and *.gif are treated the same.

Element Hierarchy

```
<config>
  <features>
```

```
<mediafiles>
  <validext>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	A comma delimited list that specifies file extensions allowed for upload.

Example

```
<validext>gif,jpg,png,jpeg,jpe</validext>
```

Maxsizek Element

Description

Specifies the maximum file size in kilobytes allowed for upload.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <maxsizek>
```

Attributes

Name	Attribute Type	Default	Description
#text	Integer	0	An integer value specifying the maximum number of kilobytes in the file. If the value is zero, the file has no size limit.

Mediaconfig Element

Description

Controls the operation of the configuration dialogs.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <mediaconfig>
```

Attributes

Name	Attribute Type	Default	Description
enabled	boolean	true	<p>Defines whether the feature is enabled. If set to false,</p> <ul style="list-style-type: none"> • login and configuration dialogs are not available to the user • the Options button is removed from the image selection dialog
allowedit	boolean	true	<p>Determines user access to the upload configuration information. This is all the connection information, except the login data.</p> <p>If this is false, the Advanced button is removed from the login dialog.</p>

Example

```
<mediaconfig allowedit="true" />
```

Transport Element

Description

Defines the mechanism used to select and upload media files.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
```

Child Elements

autoupload, username, password, port, domain, xferdir, webroot, resolvemethod, proxyserver, defsource

Attributes

Name	Attribute Type	Default	Description
Enabled	Boolean	true	If set to “ False ”, all transport values are blank.
Type	String	“FTP”	Specifies the upload mechanism to use. Values handled internally are “ FTP ” and “ FILE ”. Any other values are passed to the client application or script. The case is maintained. See Also: “Example 3: FTP” on page 466
Pasv	Boolean	True	Specifies whether the passive bit is set for FTP.
Allowupload	Boolean	True	Specifies whether a user can upload files. If true, users should only be allowed to select files already on the server. The upload mechanism (such as an external page) must use this value to prevent users from uploading files.
Xfer	String	“binary”	Specifies the low level transfer option for FTP.

Autoupload Element

Description

Defines how the automatic image upload mechanism operates. The value of the `type` attribute can specify an internal mechanism or a receiving page. (This is similar to the `type` attribute of the `transport` element.) The `openaccess` attribute determines whether login values are used for access to the remote system with the automatic upload mechanism. Finally, the `resplvl` attribute determines which automatic upload information to display to the user.

See Also: [“Automatic Upload” on page 520](#)

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <autoupload>
```

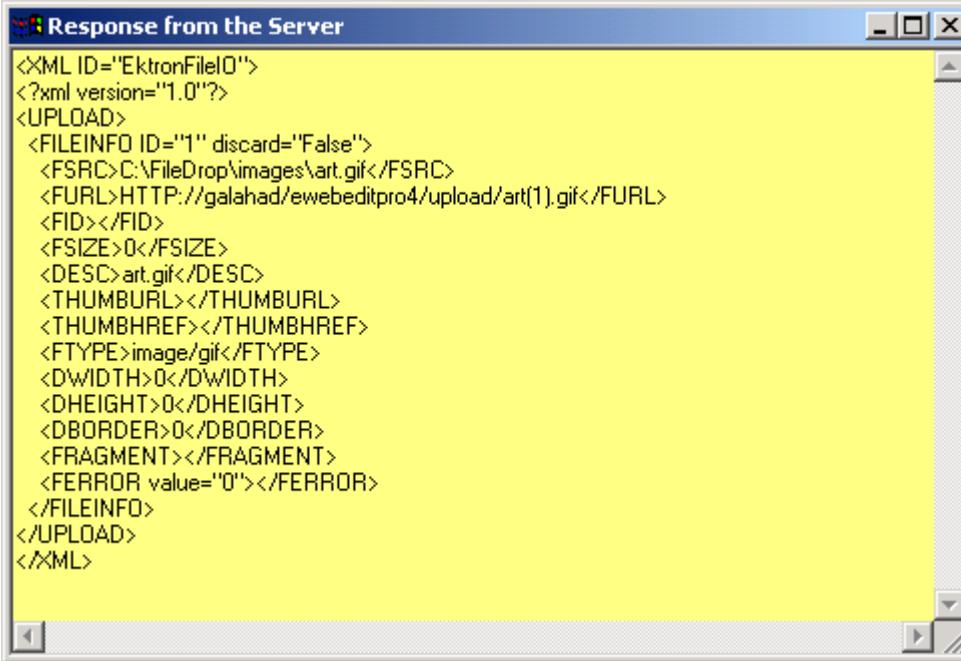
Attributes

Name	Attribute Type	Default	Description
Type	boolean	[eWebEditProPath] /ewepreceive.asp	The internal values are: FTP - Use FTP for automatic transfer NONE - No automatic upload process A receiving page address - Upload through a form post to the server. The address is case sensitive. By default, the value of the <code>transport</code> element's <code>type</code> attribute is used.
enabled	boolean	true	Determines if these settings are active. Set to true to activate them.
openaccess	boolean	true	False means use the login name, if given, for the connection in the automatic upload. True means to not use the login name in the automatic upload. This attribute exists because a login may be needed for the standard upload (older file selection) but not for the automatic upload. So, you can specify a password for the older mechanism and set this to true so that that password is not used when uploading via ASP. <i>Exception:</i> An FTP automatic upload does not read this attribute. Instead, the given login name and password are always used, if specified.

Name	Attribute Type	Default	Description
resplvl	integer	0	<p>This attribute helps a developer assemble the automatic upload receipt page by displaying automatic upload information to the user. This information, which includes error messages, server response, and process information, appears in pop-up dialogs.</p> <p>Information is displayed increasingly, with each level adding information to the previous level.</p> <p>Example:</p> <pre><autoupload type="[eWebEditProPath]/ ewepreceive.asp" resplvl="2" /></pre> <p>Use a numeric value (described below) to determine the amount of information displayed to the user.</p> <p>0 - No detailed information displayed 1 - Detailed error descriptions, if an error occurs 2 - Level 1 plus server side response information (see "Example of Automatic Upload Information Screen (Level 2)" on page 501.) 3 - Level 2 plus detailed information on each step of the upload process</p> <hr/> <p>Any value higher than 3 acts as level 3.</p>
uploadonsave	boolean	true	<p>If this is false, the automatic upload process does not occur when a user saves content.</p> <p>In this case, the <code>cmdmuuploadall</code> command must be sent either through the user's toolbar or through client scripting.</p> <p>See Also: "cmdmuuploadall Command" on page 523</p>
showdlg	boolean	true	If true , a status dialog appears while the upload process occurs.

Name	Attribute Type	Default	Description
showlistonsave	boolean	false	<p>This attribute is only in effect if the <code>uploadonsave</code> value is true. If this attribute is set to false, the list of waiting files does not appear when the content is saved. Instead, only an upload confirmation message appears.</p> <p>If this is true, a list of waiting files appears.</p>
AllowUpload	boolean	true	<p>Offers a complete override of automatic upload functionality. If this attribute is set to false, the automatic upload feature is disabled.</p> <p>If you set this to false, an error is generated if the user tries to upload. To avoid this, set the <code>TransferMethod</code> property to None to disable the upload. See Also: "Property: TransferMethod" on page 157.</p>

Example of Automatic Upload Information Screen (Level 2)



```

<?XML ID="EktronFile0"?>
<?xml version='1.0'?>
<UPLOAD>
  <FILEINFO ID='1' discard='False'>
    <FSRC>C:\FileDrop\images\art.gif</FSRC>
    <FURL>HTTP://galahad/eWebEditPro4/upload/art(1).gif</FURL>
    <FID></FID>
    <FSIZE>0</FSIZE>
    <DESC>art.gif</DESC>
    <THUMBURL></THUMBURL>
    <THUMBREF></THUMBREF>
    <FTYPE>image/gif</FTYPE>
    <DWIDTH>0</DWIDTH>
    <DHEIGHT>0</DHEIGHT>
    <DBORDER>0</DBORDER>
    <FRAGMENT></FRAGMENT>
    <FERROR value='0'></FERROR>
  </FILEINFO>
</UPLOAD>
</?XML>

```

Username Element

Description

Provides the user name for gaining access to the server. This value can be encrypted using the Ektron encryption software. Decryption is done using the licensing key provided to the editor.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <username>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The user name. Since not all external mechanisms required login access for uploading, this is optional.
Encrypted	Boolean	True	If "true", the value contained in #text is encrypted and will be decrypted before it is used.

Password Element

Description

Provides the password for gaining access to the server. This value can be encrypted using the Ektron encryption software. Decryption is done using the licensing key provided to the editor.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <password>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The password. Since not all external mechanisms required login access for uploading, this is optional.
Encrypted	Boolean	True	If "true," the value contained in #text is encrypted and will be decrypted before it is used.

Proxyserver Element

Description

Specifies the proxy server to use. Normally, this value is for FTP only.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <proxyserver>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The server name or TCP/IP address. Proxy servers are not always required.

Domain Element

Description

The domain name for the connection.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <domain>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	""	The domain name or TCP/IP address. If blank, the editor will try to determine the current domain. External mechanisms do not require a domain name, but can use one if needed.

Xferdir Element

Description

The destination directory on the server for the upload. When referenced from FTP, this is a different location than when referenced from the Web. For example, when referenced from the Web, the path might be ..\dir1\dir2\image.gif or, as a full path, /topdir/ftp/dir1/dir2. In contrast, when referenced from FTP, the path might be /dir1/dir2/image.gif.

For ASP, Cold Fusion, JSP, and other external mechanisms, the references are the same.

If the upload location and the reference location are the same, leave the webroot element blank. It will inherit the value from xferdir.

See Also: “Property: baseURL” on page 150

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <xferdir>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The destination directory for uploaded files. The case is maintained.
svrlocaleref	String	xferDispName	The locale code of the FTP the Root folder's display name. <i>See Also:</i> "Modifying the Language of eWebEditPro+XML" on page 265

Webroot Element

Description

Specifies the path to use when referencing an uploaded file.

If the server/domain is different from the upload server/domain, this value must contain the new domain, such as:

`http://www.yahoo.com/images`

If the webroot has no value, it inherits the value of the `xferdir` element.

If the Web reference domain is different from the transfer domain, the domain name must be included in the webroot element.

NOTE If you enter the domain in the webroot element, you must include the protocol. For example `HTTP://www.mydomain.com/public/pages`.

See Also: "Property: baseURL" on page 150

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <webroot>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The reference location for uploaded files. The case is maintained. If not included or blank, the value of xferdir is used.

Defsource Element

Description

This element specifies the default folder that appears when a user browses for a file on a local system. The path given can be anywhere on the local drive or a network server.

Normally, this value is used by FTP upload to help select a local file. An external selection mechanism can also use this value to specify where to retrieve a list of files.

See Also: “Setting up an Image Repository” on page 510

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <defsource>
```

Attributes

Name	Attribute Type	Default	Description
src	String	""	The location to start browsing for files to upload. The case is maintained.

Port Element

Description

Specifies which port to use for any file transfers. This value is only required if a non-standard port is used. If the value is zero or is not included, the editor determines the correct port to use.

Element Hierarchy

```
<config>
  <features>
```

```
<mediafiles>
  <transport>
    <port>
```

Attributes

Name	Attribute Type	Default	Description
#text	String	0	The port to use for file transfers. If not given or set to zero, the editor determines which port to use based on the selected protocol.

Resolvemethod Element

Description

Defines how to resolve file paths. Paths are resolved relative to the base URL (that is, the current page location).

Method	Resolves	Example
FULL	All path names to include the protocol, domain, and full path. This method ensures that paths are correct regardless of where they are referenced from.	http://www.yahoo.com/pages/images/me.gif
HOST	Relative to the root of the host server. This method lets you move directory structures to a publishing server without having to change any paths.	/pages/images/me.gif
LOCAL	Relative to the current location. This method lets you move directory structures up and down within file systems as well as to other servers.	./images/me.gif
GIVEN	To a given future location. This method resolves the paths to the location where the files will be moved. The resolved path is similar to local.	../publish/images/me.gif

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <transport>
        <resolvemethod>
```

Attributes

Name	Attribute Type	Default	Description
Value	String	LOCAL	The resolve method to use when resolving paths. The valid values are FULL , HOST , LOCAL , and GIVEN .
Src		""	The path to use for the GIVEN resolution. Case is maintained.
Allowoverride	Boolean	False	If set to “True”, the user can disable the path resolution mechanism. If disabled, paths entered by the user are not modified.
Resolve	Boolean	True	If “True”, the path resolution mechanism is enabled and will resolve paths according to the specified mechanism. If disabled, paths entered by the user are not modified.

Imageedit element

Description

Defines WebImageFX. For more information, see "WebImageFX" on page 577.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <imageedit>
```

Child Elements

control

Control Element

Description

Sets the location of WebImageFX's configuration data file.

Element Hierarchy

```
<config>
  <features>
    <mediafiles>
      <imageedit>
```

<control>

Attributes

Name	Attribute Type	Default	Description
src	String	[WebImageFXPath] / ImageEditConfig.xml	The location of WebImageFX's configuration data file.

Setting up an Image Repository

eWebEditPro+XML lets you set up an image repository folder on an intranet. Keeping all images in a central location makes it easy for users to select an image and insert it into their Web content.

The Image Repository Folder

You should create an image repository folder on a server that is accessible to client PCs, either through a UNC path or a mapped drive. (You can test this through Windows Explorer).

Next, specify the pathway to that folder in the `xferdir` element of the `transport` element in the `mediafiles` feature. You *must* enter the full path to the folder -- relative paths are not allowed. Below are two examples:

```
<xferdir src="\\imageserver\GIFS" />  
<xferdir src="M:\images\GIFS" />
```

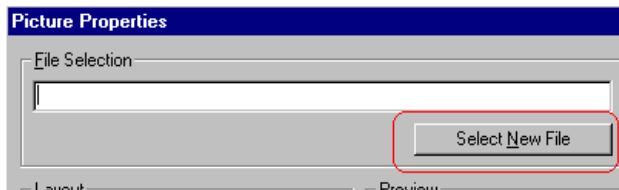
Also, set the `transport` element's `type` attribute to "`file`".

```
<transport enabled="true" type="file" xfer="binary" pasv="true">
```

WARNING!

You can only use the FILE transfer type in an Intranet setting.

Finally, you can set up a default repository folder accessible through your Intranet. If you do, your repository appears as the default folder when the user clicks **Select New File** from the Picture Properties dialog box (illustrated below).



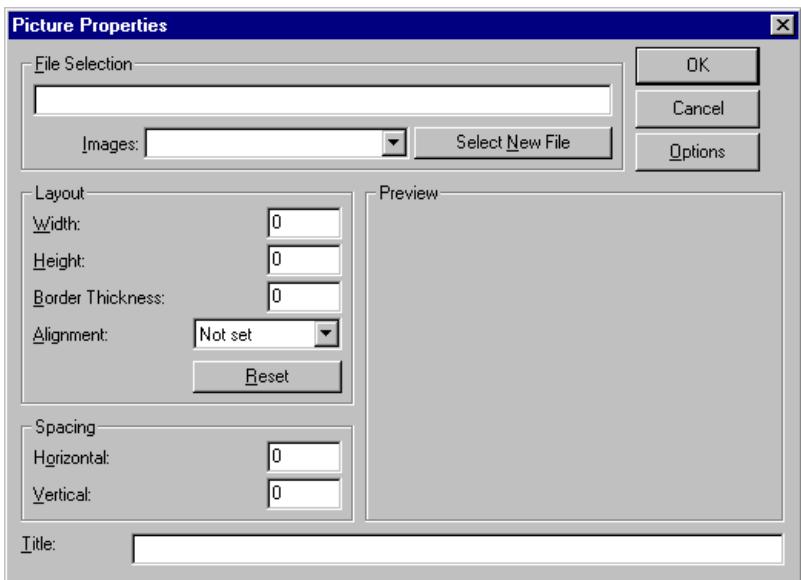
To do this, use the `defsource` element located below the `transport` element in the `mediafiles` feature. In the `defsource` element's `src` attribute, assign a local or UNC address to the `src` attribute. For example:

```
<defsource src="\\filesrvr\images\gifs" />
```

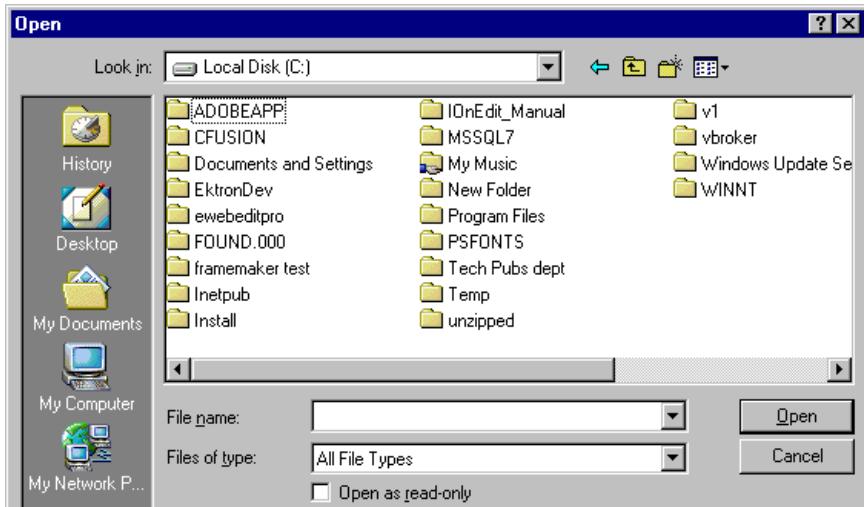
`xferdir`, `defsource` and `type` are the only elements under `transport` that you need to define. You do not need to define any other elements, such as `webroot`, `password`, `domain`, or `resolvemethod`.

Inserting an Image into a Web Page

To insert an image, the user clicks the insert picture button (). Next, the Picture Properties dialog box appears.



Here, the user clicks **Select New File** and navigates to the folder containing the image, using the standard Windows file selection dialog box.



If you set up a default directory (as explained above) that directory appears first. The user can select an image from this directory or browse to a different directory.

If a user selects an image from the image repository folder, **eWebEditPro+XML** references the image in the Web document.

If a user selects an image that is not in the image repository folder, **eWebEditPro+XML** copies the image to that folder. The Web document references the image from the folder. From then on, all users with access to the folder can insert the image.

NOTE	You cannot copy a file to the image repository folder if a file of the same name already resides there.
-------------	---

When the user inserts an image, the full path to the image is saved with the Web document. For example, if you insert an image named button.gif into your Web document, the HTML code for that line might look like this.

```
<src="file:///M:/images/GIFS/button.gif"/>
```

No other resolution options are available to the “**file**” upload type.

Example

Below is a full example of the **mediafiles** section that defines the file upload method. The **transport** element defines the upload method. All other sections are for the general feature definition.

```
<mediafiles>
    <transport type="FILE">
        <xferdir src="M:\images\GIFS" />
        <defsource src="\\filesrvr\images\gifs" />
    </transport>
    <command name="cmdmfumedia" enabled="true">
        <image key="picture"/>
        <caption localeRef="btnCapPic"/>
        <tooltiptext localeRef="btnTipPic"/>
    </command>
    <maxsizek>0</maxsizek>
    <validext>gif,jpg,png,jpeg,jif</validext>
    <mediaconfig enabled="true" allowededit="true" />
</mediafiles>
```

Dynamically Selecting Upload Destinations

When an image file is uploaded, it is moved to an upload directory defined in the configuration data. Often, you need to change the directory, depending on user interaction or other changing conditions. You can use client scripting to modify the upload directory at almost any time.

This section describes how to use scripting to change the image file upload location. The examples in this section illustrate how to change the image upload directory.

In the first example, an external upload mechanism is assigned. In Web browsers, this is an external page that contains the functionality for uploading files. The ASP sample installed with **eWebEditPro+XML** is used here.

In the second example, the upload directory is assigned and modified. The user is presented with three choices of a directory and can change it at any time. (Three choices is not a limit of the editor -- it is just a number used in this example.) The image files are uploaded, and the path information is stored in the database. The user can then select and view the files anywhere in the editor.

NOTE

The upload location, as well as other settings, can only be changed during or after the "ready" notification. This notification generates a call to the **eWebEditProReady** function, which the site administrator creates. If changes to the upload location are made before this notification, the settings are replaced by the values in the configuration data.

Setting Up Image Upload

The uploading of image files and other files is controlled through the configuration data. This data includes an upload mechanism, an upload destination directory, referencing information, and other information.

Here is an example configuration for the internal FTP image upload mechanism.

```
<mediafiles>
  <transport type="FTP">
    <domain>ftp.mydomain.com</domain>
    <xferdir src="/pages/[eWebEditProPath]/upload"/>
    <webroot src="http://www.mydomain.com/[eWebEditProPath]/
      upload"/>
  </transport>
  <command name="cmdmfumedia" style="icon">
    <image key="Picture"/>
    <caption localeRef="btnTxtPic">Picture</caption>
    <toolTipText localeRef="btnPic">Insert Picture</toolTipText>
  </command>
```

```
<validext>gif,jpg,png,jpeg,jif</validext>
</mediafiles>
```

Media File Object

The Media File Object is the interface to a selected file's internet properties. Effective use of the Media File Object is the key to manipulating the upload mechanism. The object contains information on uploading, referencing, and displaying the file.

To retrieve the interface to the object, using the `MediaFile` method.

```
var objMedia =
eWebEditPro.instances[sEditorName].editor.MediaFile();
```

or

```
var objMedia =
eWebEditPro.instances[sEditorName].editor.MediaFile();
```

Use these Media File Object methods to affect upload information.

```
getProperty(PropertyName as string, vData as Variant)
getPropertyString(PropertyName as string) as Variant
```

Use these properties to affect the upload mechanism, location, and reference.

```
TransferMethod as String - Method of Upload
DefDestinationDir as String - Upload path location
(TransferRoot as String - alias for DefDestinationDir)
WebRoot as String - Path reference from a Web page
```

WebRoot only inherits the value of DefDestinationDir when it is not assigned a value anywhere, including in the configuration data. As a result, ASP, Cold Fusion, and other scripting that use the same domain and directory structure for transfers and referencing can work with the single DefDestinationDir property.

See Also: ["Media File Object" on page 20](#)

Modifying the Upload Location

Configuration Data

Any customization should begin with the configuration data, since all default settings are defined there. In this case, the `mediafiles` section is the focus. The following example explains how to address these issues.

- Since the example changes the upload mechanism programmatically, it defaults to the internal FTP setting.
- The default upload destination directory is defined in the configuration file.
- Since the Web reference path is the same as the destination, the example does not include a `webroot` element. As a result, the `WebRoot` property inherits the value defined in the `xferdir` element and any value assigned to the `DefDestinationDir` property.

Here is the `mediafiles` section of the configuration file for this example.

```
<mediafiles>
  <transport type="FTP">
    <domain></domain> <!-- empty means use current -->
    <xferdir src="[eWebEditProPath]/upload"/>
    <resolvemethod value="local"/>
  </transport>
  <command name="cmdmfumedia" style="icon">
    <image key="Picture"/>
    <caption localeRef="btnTxtPic">Picture</caption>
    <toolTipText localeRef="btnPic">Insert Picture</toolTipText>
  </command>
  <validext>gif,jpg,png,jpeg,jif</validext>
</mediafiles>
```

Sample HTML Page

Here is sample HTML source page to start with. It is a simple page that contains the **eWebEditPro+XML** editor, a list selection item, and a text item to display the current upload path. The path manipulation functionality is added later.

```
<html>
<html>
<head>
  <title>User Select Upload Location</title>
  <script language="JavaScript1.2" src="ewebeditpro.js"></script>
</head>
<body>
<form method="post" name="selectpath">
<h1>User Select Upload Location</h1>
<script language="JavaScript1.2">
var g_strUserEditorName = "MyContent1";
document.write('<input type=hidden name=' + g_strUserEditorName +
  '" value="This is initial content.">');
if (typeof eWebEditPro == "object")
{
  eWebEditPro.create(g_strUserEditorName, "100%", 400);
}
</script>
</form>
<p><b>Current Upload Location:</b><br>
<select name="LocationSel" title="Current Upload Location"
OnChange="UseSelectedLocation(LocationSel.selectedIndex)">
<option>Configured Location<option>Work Location
<option>Publish Location</select><b>:&nbsp;</b>
<input name="CurUploadLocation" style="text" value="Upload Destination"
maxlength="256" size="80"></p>
</body>
</html>
```

Notice a call to the `UseSelectedLocation` function -- this is explained in "User Selection – Changing the Upload Location" on page 516.

Initialization

When the page loads, an external upload mechanism is assigned. The code also reflects to the user the current upload path from the Media File Object.

```
<script language="JavaScript1.2">
var g_strConfiguredPath = "";
function eWebEditProReady(sEditorName)
{
var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
objMedia.setProperty("TransferMethod", "samples/asp/database/mediamanager.asp");
g_strConfiguredPath = objMedia.getPropertyString("DefDestinationDir");
document.selectpath.CurUploadLocation.value = g_strConfiguredPath;
}
</script>
```

The current path (the default path defined in the configuration file) is stored for later use. In our example, this allows the user to re-select it.

Note on the Missing `eWebEditProReady`

To the core JavaScript files installed with **eWebEditPro+XML**, `eWebEditProReady` is a reserved function name. This function does not exist in the core JavaScript files. Instead, a developer must define the function either in his/her own JavaScript files or in the HTML file.

The function can be defined in any file brought in by the page. In the example below, this function is defined in the HTML file.

When the editor control sends the "ready" notification, the core JavaScript checks to see if `eWebEditProReady` is defined. If it is, the core JavaScript calls the function to notify the scripts that the editor is ready.

User Selection – Changing the Upload Location

In this example, the user can select one of three image upload locations. Here is the JavaScript to handle the selection:

```
function AssignUploadLocation(sEditorName, sLocation)
{
    var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
    objMedia.setProperty("DefDestinationDir", sLocation);
    document.selectpath.CurUploadLocation.value =
        objMedia.getPropertyString("DefDestinationDir");
}
function UseSelectedLocation(iIndex)
{
    var UploadLoc = new Array(g_strConfiguredPath, eWebEditProPath + "samples/common/database",
        "/publish/images");
    AssignUploadLocation(g_strUserEditorName, UploadLoc[iIndex]);
}
```

The `UserSelectedLocation` function uses the index passed down from the `LocationSelection` item to specify the user's selection.

UserSelectedLocation then sends the path associated with that selection to the AssignUploadLocation function.

The important function to examine is AssignUploadLocation, which illustrates how to set the destination directory for any uploads. This function receives the location and sets it into the Media File Object.

Full Example

Here is the full HTML page that shows how to change the upload location.

```

<html>
<head>
    <title>User Select Upload Location</title>
    <script language="JavaScript1.2" src="ewebeditpro.js"></script>
</head>
<body>
<script language="JavaScript1.2">
var g_strConfiguredPath = "";
function eWebEditProReady(sEditorName)
{
    var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
    objMedia.setProperty("TransferMethod",
        "samples/asp/database/mediamanager.asp");
    g_strConfiguredPath =
        objMedia.getPropertyString("DefDestinationDir");
    document.selectpath.CurUploadLocation.value = g_strConfiguredPath;
}
function AssignUploadLocation(sEditorName, sLocation)
{
    var objMedia = eWebEditPro.instances[sEditorName].editor.MediaFile();
    objMedia.setProperty("DefDestinationDir", sLocation);
    document.selectpath.CurUploadLocation.value =
        objMedia.getPropertyString("DefDestinationDir");
}
function UseSelectedLocation(iIndex)
{
    var UploadLoc = new Array(g_strConfiguredPath,
        eWebEditProPath + "samples/common/database",
        "/publish/images");
    AssignUploadLocation(g_strUserEditorName,
        UploadLoc[iIndex]);
}
</script>
<form method="post" name="selectpath">
<h1>User Select Upload Location</h1>
<script language="JavaScript1.2">
var g_strUserEditorName = "MyContent1";
document.write('<input type=hidden name="' + g_strUserEditorName +
    '" value="This is initial content.">');
if (typeof eWebEditPro == "object")
{
    eWebEditPro.create(g_strUserEditorName, "100%", 400);
}
</script>
<p><b>Current Upload Location:</b><br>
<select name="LocationSel" title="Current Upload Location"
    OnChange="UseSelectedLocation(LocationSel.selectedIndex)">
    <option>Configured Location</option>
    <option>Work Location</option>
    <option>Publish Location</option>
</select><b>&nbsp;&nbsp;</b>
<input name="CurUploadLocation" style="text" value="Upload Destination"
    maxlength="256" size="80"></p>
</form>
</body>
</html>

```

In the above example, the two important script functions are `eWebEditProReady` and `AssignUploadLocation`. These functions show how to access and modify the upload method and location.

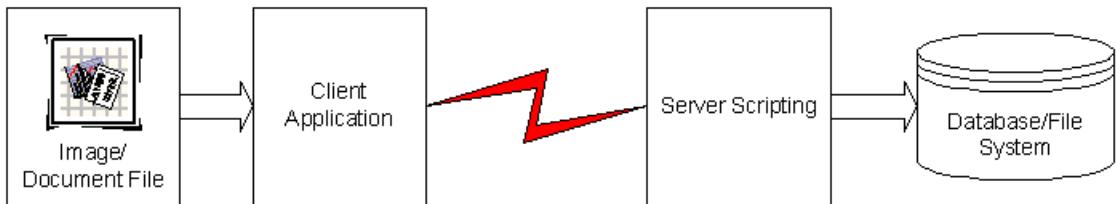
Automatic Upload

NOTE

The Automatic Upload feature is not supported in Microsoft IE4 or earlier browsers, nor is it supported for Microsoft Word 97.

There are two kinds of Automatic Upload:

- content containing local files and images linked to an external application, such as MS Word. When the user saves the content, the files and images are converted to HTML. This is described in this chapter.
- content is uploaded directly to the server without leaving the editor. This is described in "[Content Upload](#)" on page 563.



Automatic Upload of Files and Images from an External Application

This type of automatic upload occurs when a user inserts content into **eWebEditPro+XML** that contains local files and images that are linked to an external application, such as MS Word. When the user saves the content, the files and images are converted to HTML.

Next, the user confirms the upload to the server through a dialog box that lets him select which files to upload and which to leave as local until the content is completed. Finally, the editor uploads the files to the server, which stores them in a file system. After a file is uploaded to the server, its path changes from local to the destination defined in the `xferdir src` attribute of the configuration data.

Automatic upload is complicated because physical drives need conversion to URL values, and files may change their names when uploaded. Also, issues of extended file processing and how to represent an uploaded file in the content must be resolved.

The Automatic Upload feature sets up two-way communication between client and server. The responding XML data provides the information needed to resolve most issues. As a result, the client editor can know what happens to the uploaded file and how to use the file.

This kind of Automatic Upload includes these capabilities:

- Simple Path Resolution

- Specific File Types
- Appropriate Content Modification
- Client Notification
- Resolve File Processing Changes
- Upload Status (server side of status information)
- Security

To provide these capabilities, Automatic Upload uses previously available mechanisms. For example, the client side functionality uses the available API within Microsoft's Wininet DLL.

Also, since Automatic Upload is an extension of the image processing feature, you can use the configuration data described in ["The Mediafiles Feature" on page 493](#) to determine many aspects of the feature. For example, you can use the configuration data to determine the

- Upload mechanism
- Server connection and active page
- Security and login
- Upload limits

This feature is explained more fully in the following topics.

- [Installing the Automatic Upload Feature](#)
- [Modules that Enable Automatic Upload](#)
- [cmdmfuploadall Command](#)
- [Overview of the Automatic Upload Process](#)
- [Information Components](#)
- [eWebEditPro Fields Sent with Post](#)
- [Creating an Automatic File Receive Script](#)
- [Steps to Receiving a File](#)
- [Data Island](#)
- [EWepAutoSvr Object API](#)
- [EkFileObject API](#)
- [XML Element Descriptions](#)
- [ColdFusion example](#)
- [ASP example](#)
- [AutomaticUpload Object](#)

Installing the Automatic Upload Feature

During the server installation of **eWebEditPro+XML**, the following question appears:

Do you want to register the Ektron automatic file upload component? If not, you will need to use another mechanism to upload images.

You must click **Yes** to use the features described in this section.

Modules that Enable Automatic Upload

The feature contains two files to process the file transfer.

File	Resides on the	Description
eWepMediaTransfer.dll	client	The module that the editor uses to initiate a transfer. This object assembles form data and posts it to the server. The accepting page written to accept files is also specified on the client.
eWepAutoSvr.dll	server	ASP module that extracts and saves the uploaded file. Its design is primarily for ASP. If a server-side script supports COM on a PC, you can use this object to help accept the file. The server script must allow the retrieval of the raw posted form data for this object to work.

The feature also includes *ewepreceive.asp*, a default ASP script that uses the *eWepAutoSvr.dll* module and controls the upload if no other receiving page is available.

For instructions on how to write a receiving script, see ["Creating an Automatic File Receive Script" on page 529](#).

An Example of Customizing Automatic Upload

The ASP database sample illustrates how to customize the Automatic Upload mechanism to update a database. Specifically

- the *mediaautoreceive.asp* file receives the file and updates the database
- the *edit.asp* file programmatically changes the editor's upload receiving page when a "ready" notification is received

To try out the ASP database sample, follow this path, beginning with the Windows Start button:

Start > Programs > Ektron > eWebEditPro3 > Samples > ASP > Database > ASP Database Sample

The mediaautoreceive.asp and edit.asp files are installed to `eWebEditPro path/samples/asp/database`.

cmdmfuuploadall Command

The cmdmfuuploadall command lets the user manually perform the upload process. The command displays the Files Waiting for Upload dialog box, which prompts the user to upload files.

This command is not on a standard toolbar in the default configuration, but can be added to a toolbar when customization is enabled.

See Also: ["Adding a Toolbar Button" on page 237](#)

The Automatic Upload feature also uses the transport element (see ["Transport Element" on page 497](#)) and the AutomaticUpload Object (see ["Automatic Upload Object" on page 562](#)).

Overview of the Automatic Upload Process

When content containing files and links to files is copied from an application (such as MS Word) to the editor, the user may want to upload the files and update the paths before saving the content. The Automatic Upload mechanism provides this capability.

If the client knows the correct server page and the server contains that page, the user needs to know nothing about the upload process, except maybe his or her login. The upload happens automatically.

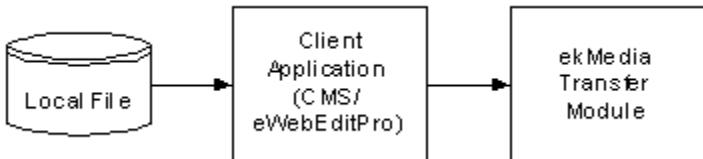
The Upload Process

Automatic Upload allows an application, such as Ektron **eWebEditPro+XML** or CMS300, to easily upload files to the server. The process starts on the client side application.

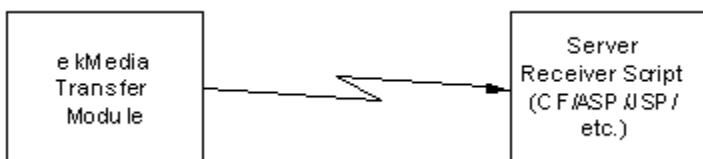
1. One of these events initiates the upload process:
 - The user saves the editor content
 - The cmdmfuuploadall command is available as a toolbar button, and the user presses the button

See Also: ["cmdmfuuploadall Command" on page 523](#)
2. The content is scanned, and files that meet these criteria are given to the eWepMediaTransfer module for uploading.
 - The file is specified in the body
 - The file path uses an `href` or `src` attribute
 - The file exists on the local system

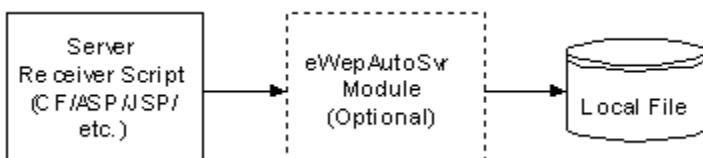
See Also: ["Modules that Enable Automatic Upload" on page 522](#)



3. The eWepMediaTransfer module posts the file information, and any other information given to it by the client, to the server as multipart form data. A server script receives the posted information.
The form post consists of a posted form with a file field. Form data is assembled on the client side. The form contains information fields and a file field. This is the same format found in any form being posted from a browser.
See Also: "[eWebEditPro+XML Fields Sent with Post](#)" on page 526

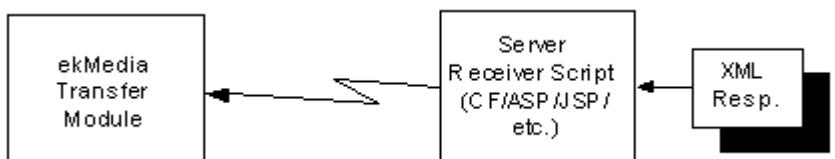


4. The server script extracts the file and information about it, and updates the content management system with this information. This involves saving the file and updating any databases or files.
Other file processing can also be done, such as creating a thumbnail or specifying an icon to represent the uploaded file. An ASP script is provided in the eWepAutoSrv module to help this process.



See Also: "[Modules that Enable Automatic Upload](#)" on page 522; "[Assembling the Response XML Data Island](#)" on page 537

5. When the file processing is complete, the server assembles XML response data and sends it to the client. The eWepAutoSrv file is provided for ASP or other scripts that support COM to help this process.



6. When the XML data is received, the eWepMediaTransfer module parses it and makes it available to the client application. The data includes transfer status, processing information, and other information. Also, the client can extract the response data for its own parsing. The client uses the information to determine how to represent the uploaded file.

Information Components

The following information components are used to maximize the processing of file data within edited content.

Component	Description
Final full file URL	The server decides where the file is finally placed and what its name is. This is also the resulting file if the uploaded file is converted to another format. For example, a Word document is converted to a PDF file. This must be the full reference used in a browser.
Resulting Title	A title is sent with the upload, but the server may want to change it. An example is “translations”.
Referenced File Type	May have been changed. For example, a Word document may be converted to a PDF file.
Thumbnail File	A URL to a thumbnail created for the file.
Thumbnail Link	The URL link for when the thumbnail is clicked.
Error Description	This can include a number, a description, and/or a suggested course of action.
Discard Reference	If a file can be uploaded but the user should not reference it, this instructs the editor to omit the file's reference from the content.

Concepts

These are general concepts behind this data.

- The data can include more than one file. While the client may not support sending more than one file in a transmission, the server side must be ready for this extension.
- If a thumbnail is specified, it is used in place of the image reference.
- If a thumbnail is specified but no href value is assigned, the image URL is used as the link for the thumbnail image.

- If this information is not sent to the client, the client editor “guesses” where the file ended up from information contained in the configuration data.

eWebEditPro+XML Fields Sent with Post

The **eWebEditPro+XML** client sends one of these sets of fields with a file post.

- Image Upload Fields
- Custom Field Set

Image Upload Fields

These fields send information about an image file. They are also sent with the ASP sample provided with **eWebEditPro+XML**. This set of fields provides compatibility with the ASP database sample and lets the server side receive a large amount of data about the file.

Much of the sent data originates in **eWebEditPro+XML**'s configuration data stream. The server-side component can extract what it needs and ignore the other fields.

Entry	Description
actiontyp	<p>The action type. (Note that the final 'e' is missing.) This entry's value is a command to the server. It is normally <code>uploadfile</code> for the file upload process.</p> <p>See Also: "uploadfile" on page 531</p>
editor_media_path	<p>The requested upload destination. For example, <code>http://www.mysite.com/uploads</code>.</p> <p>Normally, this value is equal to the <code>web_media_path</code> value. It is given here for if it is different for some reason.</p> <p>The server can ignore this and place the file where it wants. If the server places the file elsewhere, the client should receive the full XML response so that it knows where the file was placed.</p>
ekclientname	<p>The name of the client application. It is not necessarily the application name, although you can use the application name.</p> <p>This is usually the <code>eWepMediaTransfer</code> value. Error check this.</p>

Entry	Description
ekclientneeds	<p>The level of response the client requests. An application may want to receive only the file name, all the data, or nothing. The values are:</p> <p>fullxml - full XML data about the upload filename - just the file name, no XML data none - no data returned</p> <p>The default is "filename." This chapter describes how to generate code for the fullxml value.</p>
ekclientversion	<p>The version of the upload module on the client side. This allows the server side to determine what the client supports. The version <i>must</i> be 1.0 or higher to support the upload mechanism. Error check this.</p> <p>If no version is sent, this field returns the file name only. Backward compatibility is always assumed, so if the client version is higher than the server component, the server component uses the highest version format that it supports.</p>
extension_id	<p>The file extension given as an ID. This can be used to categorize the file in a database.</p> <p>The ASP sample processes this value into an ID number. The client editor will echo this operation.</p> <p>Most likely, the server side wants to determine the ID.</p>
extensions	<p>The list of valid extensions contained in the configuration XML data. An example value is: "gif,jpg,png,jpeg,jif"</p> <p>The receiving client should examine these extensions to ensure the file being uploaded is acceptable. If the file extension is not acceptable, set the 'discard' attribute of the 'FILEINFO' element to true.</p> <p><i>See Also:</i> "discard" on page 533</p>
filename	<p>The name of the local file. Normally, this is a full local path that does not match a file location on the server. An example is: C:\Inetpub\wwwroot\kewl.gif</p>

Entry	Description
file_size	<p>The file size in bytes. This is meant for an early determination of validity before processing is done. Once a file is uploaded and saved, the EWebsAutoSrv module determines the size from the file. This second value is used in the response.</p> <p>See Also: "Assembling the Response XML Data Island" on page 537</p>
file_title	<p>The file's title or description. This value is used in the alt and title attributes of an image file. The server can override this value using the Description method.</p> <p>See Also: "Description" on page 545</p>
file_type	<p>The type of file. This follows the HTML convention where a GIF file is an "image/gif" type.</p>
height	<p>The requested height to show the image. The server can override this using the FileDimensions method.</p> <p>See Also: "FileDimensions" on page 545</p>
img_date	<p>The date of the file. The format is "11/30/2002 10:33:51 PM".</p> <p>This is <i>not</i> the date of the upload. The upload date is sent with the ASP sample, but the file date is more useful, since the upload date can be determined on the server side.</p>
uploadfilephoto	<p>The file selection field.</p>
web_media_path	<p>The expected reference location, such as <code>http://www.mysite.com/uploads</code>.</p> <p>This location is assumed if the server does not respond with a reference location value. The server uses the FileUrl method to specify where the location can be referenced.</p> <p>The XML response element that specifies this is FURL.</p> <p>See Also: "FURL" on page 555</p>
width	<p>The requested width to show the image. The server can override this using the FileDimensions method.</p> <p>See Also: "FileDimensions" on page 545</p>

Custom Field Set

A client application, such as Ektron CMS300, can add fields to the posted form. As you extend the receipt script functionality, you can look for any custom fields that you know about and act on their data.

Example HTML Form

The following HTML example is provided to help you understand how Automatic Upload's posted form fields might look like if defined them as HTML source. An HTML file is *not* used for the automatic upload, but the example illustrates the fields that a receiving script would expect.

```

<html>
<!-- This is never used for an automatic upload, but from what the server sees
     it is as if a page like this were being posted up to the server. -->
<head>
  <title>EktronFileIO Upload Example</title>
</head>
<body>
  <h1>Upload a File Using eWepAutoSvr</h1>
  <form action="/ewebeditpro5/ewepreceive.asp" method="POST" enctype="multipart/form-data"
        name="frmupload">
    <h3>These are the Fields Submitted:</h3>
    Select File: <input type="File" name="uploadfilephoto" size="20" maxlength="256"/> <br/>
    Client Name: <input type="Text" name="ekclientname" value="ekmediatransfer"/> <br/>
    Client Version: <input type="Text" name="ekclientversion" value="1.0"/> <br/>
    Response Need: <input type="Text" name="ekclientneeds" value="fullxml"/> <br/>
    Action Type: <input type="Text" name="actiontyp" value="uploadfile"/> <br/>
    Image Date: <input type="Text" name="img_date" value="1/10/2003 10:33:51 PM"/> <br/>
    Extension ID: <input type="Text" name="extension_id" value="0"/> <br/>
    File Type: <input type="Text" name="file_type" value="image"/> <br/>
    Upload Location: <input type="Text" name="editor_media_path" value="/ewebeditpro5/upload"/>
    <br/>
    URL to Use: <input type="Text" name="web_media_path" value="/ewebeditpro5/upload"/> <br/>
    Valid Exts.: <input type="Text" name="extensions" value="gif,jpg,png,jpeg"/> <br/>
    File Size: <input type="Text" name="file_size" value="4096"/> <br/>
    Width: <input type="Text" name="width" value="800"/> <br/>
    Height: <input type="Text" name="height" value="600"/> <br/>
    File Title: <input type="Text" name="file_title" value="This is a picture of me."/> <br/>

    <br/><input type="submit" name="btnupload" value="Upload File"/>

  </form>
</body>
</html>

```

Creating an Automatic File Receive Script

The Automatic Upload's server-side scripts are designed to

- save the uploaded file
- return to the client XML data about the upload

This section describes how to create a server-side script or object to receive a file from the Automatic Upload mechanism. You can use these instructions to create any server-side script, from ColdFusion to JSP to .Net. Sample scripts are not provided to explain how to perform the required operations, although an ASP sample appears in ["ASP Example" on page 559](#).

What This Section Covers

- Description of data received by the server
- How to receive a file
- Samples in ASP with eWepAutoSvr
- Description of data sent back to the client

What This Section Does Not Cover

- How to perform CMS operations, such as updating a database
- Specific server-side scripting language or object language

The Automatic Upload Server-Side Receiving Module

The automatic upload server-side receiving module (EWePAutoSvr on IIS systems) extracts the file and information about the file, and updates the content management system with this information. This involves saving the file and updating any databases or files.

The module consists of information components and a data island, which is implemented as a repository for the return data. For more information about the information components and data island, see ["Information Components" on page 525](#) and ["Data Island" on page 534](#).

Other file processing can also be done, such as creating a thumbnail or specifying an icon to represent the uploaded file. The ASP script is provided in the EWePAutoSvr module to help this process.

When the file processing is complete, the server assembles XML response data and transmits it to the client. The EWePAutoSvr file is provided for ASP or other scripts that support COM to help this process. For more information on this procedure, see ["Steps to Receiving a File" on page 530](#).

Steps to Receiving a File

NOTE

To learn about receiving uploaded content, see ["Steps to Receiving Content" on page 539](#).

There are six steps to receiving a file:

1. Act on the Command
2. Extract the File Information
3. Determine the File Destination

4. Extract the File Binary and Save
5. Build the Return XML Data
6. Send it Along

Step 1 – Act on the Command

The command is retrieved from the `actiontyp` field in the posted form. The client object sends one of two known commands: `uploadcontent` and `uploadfile`.

uploadcontent

When the server receives this value, the posted form contains the document content. The server then assembles a response that is formatted in HTML, which is displayed in the editor. For more information, see "[Content Upload](#)" on page 563.

uploadfile

When the server receives this value, the posted form contains a file. The text returned is described in the rest of this section.

Unknown Commands

There is no mechanism for allowing the client to send non-standard commands to the server.

Step 2 – Extract the File Information

Information about the upload is broken down and stored within every field except the "uploadfilephoto" field, which contains the image.

You should extract this information before determining how to proceed. Error checking and the expected client response level that you specify here determine how the script should process and respond.

The other information is used for error checking, database operations, etc. Because processing this information is specific to each content management system, it is not covered here.

Step 3 – Determine the File Destination

From the file name retrieved in Step 2, determine if a file by that name already exists. If overwriting files is not allowed, the script must make the file name unique.

The script can use the requested logical upload destination or determine its own. The requested destination is within the `editor_media_path` field. Normally, this value is defined in the configuration file, but the server can determine another location for the file.

No matter what the location, its logical location is returned in the XML data. You should map the logical location to a physical location for saving the file on the server's hard drive.

Step 4 – Extract the File Binary and Save

The binary of the file exists in the `uploadfilephoto` file selection field. Extract the binary data from this field and save it to the location and name determined in Step 3.

NOTE The `eWepAutoSvr` object is mainly for ASP. Other scripting languages that support COM and can extract the submitted form as binary data can also use it.

Step 5 – Build the Return XML Data

NOTE The XML response is only required when a file is uploaded to the server. This should not be the response for a content upload.

This is the most complex section of the process. You must follow the XML format. Generally, you need to create one tag per piece of data.

Except for the xml declaration tag, all tags are upper case. Since XML is case sensitive, this convention helps distinguish the upload information tags from other XML tags in returned content.

Upload data items are assembled as content within the tags and not attributes. For example: `<FTYPE>image/gif</FTYPE>`.

For a full list of tags used in the returned XML data, see "[XML Element Descriptions](#)" on page 551.

Start with the XML Root Tag

XML data must contain a root tag. For the automatic upload feature, the root tag is the `<UPLOAD>` tag.

XML also needs the XML declaration. Since we want to support data islands, our declaration must use the HTML XML data tag.

So, here is the checklist for the root setup of the XML:

- HTML XML data tag (`<XML>`)
- XML declaration (`<?xml version="1.0"?>`)
- The 'UPLOAD' root tag (`</UPLOAD>`)

The root of the returned XML data must always be this:

```
<XML ID=EktronFileIO>
<?xml version="1.0"?>
<UPLOAD>
```

```
</UPLOAD>
</XML>
```

The upload data goes between the `<UPLOAD>` tags.

Add the File Information Tag

Information for each uploaded file is contained within the <FILEINFO> tag, which has two attributes.

FILEINFO Attribute	Description
ID	<p>An ID value assigned by the server to uniquely identify a file in the data.</p> <p>This is not the same as the ID element, which is the value assigned by the client and most likely will not match.</p>
discard	<p>If the server accepted the file but does not want it used within the content, it sets this value to <code>true</code>. The client receives the data and corrects the content to not contain the file.</p>

Each uploaded file contains one of these attributes and all the data contained within it. eWebEditPro+XML uploads only one file with each post, so normally you have one <fileinfo> entry.

With the <FILEINFO> tag, the returned XML data should look something like this:

```
<XML ID=EktronFileIO>
<?xml version="1.0"?>
<UPLOAD>
    <FILEINFO ID="0" discard="False">
        </FILEINFO>
    </UPLOAD>
</XML>
```

Adding the File Information

The rest of the elements contain data about the file.

IMPORTANT!

These elements *must* exist, even if there is no data within them.

This completes the XML data.

```
<XML ID=EktronFileIO>
<?xml version="1.0"?>
<UPLOAD>
    <FILEINFO ID="0" discard="False">
        <FSRC>C:\Inetpub\wwwroot\Arrows\next0.gif</FSRC>
        <FURL>http://www.echo.com/ewebeditpro3/upload/me(1).gif</FURL>
        <FID></FID>
        <FSIZE>128</FSIZE>
        <DESC></DESC>
        <THUMBURL></THUMBURL>
        <THUMBREF></THUMBREF>
```

```

<FTYPE>image/gif</FTYPE>
<DWIDTH>0</DWIDTH>
<DHEIGHT>0</DHEIGHT>
<DBORDER>0</DBORDER>
<FRAGMENT></FRAGMENT>
<FERROR value="0"></FERROR>
</FILEINFO>
</UPLOAD>
</XML>

```

Step 6 – Respond Back to the Client

The XML data can be sent back by itself, or as part of a larger HTML page. Normally, the bare-bones XML shown above is all that is returned, since it is all the editor looks at.

But, if you are using a CMS that displays the resulting data, you may want to return a complete HTML page. If you want to look sophisticated, include this data island on a page with a table to display the data. See Also: ["Data Island" on page 534](#)

Creating the Script

Here's a practical example of how to create the script.

1. Implement a page that logs every receipt of an HTTP post (to a text file, for example).
2. Configure WIFX to upload to this page. For example, if you create a page at the path `/postacceptor/WebForm1.aspx`, set the following in the `ImageEditConfig.xml` file.
`<autoupload type="/postacceptor/WebForm1.aspx" />`
3. Try to upload from WIFX.
4. Check the page's log to confirm that WIFX hit it. Note that WIFX returns an error even if this works properly because the page doesn't yet return the proper XML packet.
5. After confirming that WIFX is hitting the page, change it slightly to loop through all posted form fields and log their values. This lets you see exactly what information WIFX is posting.
6. Write the code needed to produce the XML packet. See Also: ["Step 5 – Build the Return XML Data" on page 532](#)
7. Set the page to return an XML packet with dummy values. These should take care of the WIFX error message and display a message like **Upload Successful!** after the form is posted.
8. Decode the MIME packet in the image files and save it to the server's hard drive.

Data Island

A data island is implemented as the repository for the return data. This is generated by the server-side component and sent back in a standard Web page.

Below is a sample data island and how it could be used in a returned page. For a description of the XML elements used in this island, see "XML Element Descriptions" on page 551.

See Also: "Assembling the Response XML Data Island" on page 537

```
<html>
<head><title>Sample Data Island</title></head>
<body>
<XML ID=EktronFileIO>
    <?xml version="1.0"?>
    <UPLOAD>
        <FILEINFO ID="0" discard="false" width="0" height="0" border="0">
            <FSRC>iraqshow.jpg</FSRC>
            <FURL>
                http://us.news1.yimg.com/us.yimg.com/p/rids/20021219/s/1040306979.3758604392.jpg
            </FURL>
            <FSIZE>102047</FSIZE>
            <DESC>Slideshow: Iraq and Saddam Hussein</DESC>
            <THUMBURL>
                http://us.news1.yimg.com/us.yimg.com/p/rids/20021219/t/1040306979.3758604392.jpg
            </THUMBURL>
            <THUMBREF>
                http://story.news.yahoo.com/news?g=events/wl/082701iraqplane&tmpl=sl&e=1
            </THUMBREF>
            <FTYPE>image/jpeg</FTYPE>
            <FERROR val="0"></FERROR>
        </FILEINFO>
        <FILEINFO ID="1">
            <FSRC>ivory_coast_xcn106.jpg</FSRC>
            <FURL>
                http://us.news2.yimg.com/us.yimg.com/p/ap/20021218/capt.1040241167.ivory_coast_xcn106.jpg
            </FURL>
            <FSIZE>102047</FSIZE>
            <DESC>Slideshow: Ivory Coast Conflict</DESC>
            <THUMBURL>
                http://us.news1.yimg.com/us.yimg.com/p/ap/20021218/thumb.1040241167.ivory_coast_xcn106.jpg
            </THUMBURL>
            <THUMBREF>
                http://story.news.yahoo.com/news?tmpl=story&u=/021218/168/2w49d.html
            </THUMBREF>
            <FTYPE>image/jpeg</FTYPE>
            <FERROR val="0"></FERROR>
        </FILEINFO>
        <FILEINFO ID="2">
            <FSRC>mdf171290.jpg</FSRC>
            <FURL>http://us.news2.yimg.com/us.yimg.com/p/nm/20021218/mdf171290.jpg</FURL>
            <FSIZE>102047</FSIZE>
            <DESC>'Rings' Leads Charge to Record</DESC>
            <THUMBURL>
                http://us.news1.yimg.com/us.yimg.com/p/nm/20021218/amdf171290.jpg
            </THUMBURL>
            <THUMBREF>
                http://story.news.yahoo.com/news?tmpl=story&u=/021218/161/2w6dq.html
            </THUMBREF>
            <FTYPE>image/jpeg</FTYPE>
            <FERROR val="0"></FERROR>
        </FILEINFO>
```

```

</UPLOAD>
</XML>
<h2>Uploaded File Information</h2>
<table DATASRC="#EktronFileIO cellpadding=3 border=1>
<thead>
<tr>
<th>Thumbnail</th>
<th>Image Display</th>
<th>Description of Image</th>
</tr>
</thead>
<tr>
<td><a DATAFLD="THUMBREF" target="_blank"><img DATAFLD="THUMBURL"></a></td>
<td><img DATAFLD="FURL"></td>
<td><div DATAFLD="DESC"></div></td>
</tr>
</table>
</body>
</html>

```

The above is a complete example of how a data island could be returned. The example produces this output.

Uploaded File Information		
Thumbnail	Image Display	Description of Image
		U.S. Navy, Iraq and Bahrain Sailors
		U.S. Navy, Iraqi Coast Guard
		U.S. Navy Change of Command

Here is a simple data island, by itself, generated by the server-side component.

```

<XML ID=EktronFileIO>
  <?xml version="1.0"?>
  <UPLOAD>
    <FILEINFO ID="0">
      <FSRC>mdf171290.jpg</FSRC>
      <FURL>http://us.news2.yimg.com/us.yimg.com/p/nm/20021218/mdf171290.jpg</FURL>
      <FID></FID>
      <FSIZE>128000</FSIZE>
      <DESC>'Rings' Leads Charge to Record</DESC>
      <THUMBURL>
        http://us.news1.yimg.com/us.yimg.com/p/nm/20021218/amdf171290.jpg
      </THUMBURL>
      <THUMBHREF>
        http://story.news.yahoo.com/news?tmpl=story&u=/021218/161/2w6dq.html
      </THUMBHREF>
      <FTYPE>image/jpeg</FTYPE>
      <DWIDTH>0</DWIDTH>
      <DHEIGHT>0</DHEIGHT>
      <DBORDER>0</DBORDER>
      <FRAGMENT></FRAGMENT>
      <FERROR val="0"></FERROR>
    </FILEINFO>
  </UPLOAD>
</XML>

```

The above data island example is simplified from the full example shown above. It is shown isolated from the rest of the HTML, and contains only one uploaded file. This example is a typical string returned from EWepAutoSvr or the other server-side scripts.

Assembling the Response XML Data Island

You can use the EWepAutoSvr module's interface to assemble the response XML data island. An object interface is used in place of a DOM interface to set the values.

The ASP script sets values in the eWepAutoSvr object. Then, the eWepAutoSvr object can produce the resulting full XML data island, which can be placed in the returning document.

See Also: "EWepAutoSvr Object API" on page 540

Example

Below is an example of using the EWepAutoSvr module to create the complex example shown in "Data Island" on page 534.

```

<!-- #include file="thumbnailmaker.asp" -->
<html>
<head><title>File Upload Response</title></head>
<body>
<h2>File Upload Response</h2>
<%
  Dim g_strDataIslandID  ' holds the ID of the response data
  Dim g_iClientMajorRev
  Dim g_iClientMinorRev
  Dim g_iFileCount

```

```

ReceiveSubmittedFiles    ' Saves the submitted files.

    Response.Write("<p>Client version is: " & g_iClientMajorRev & "." & g_iClientMinorRev &
"</p>")
    Response.Write("<p>There were " & g_iFileCount & " files uploaded.</p>")

.....
' Call the routine to save the submitted files
' to local locations.
' This also processes the uploaded files and
' generates the response data.
Sub ReceiveSubmittedFiles()
    Dim BinaryFormData, uploadObj, fileObj, ServerLocation
    Dim strNewFileName, strFileLoc, ErrorCode, iFileIdx

    BinaryFormData = Request.BinaryRead(Request.TotalBytes)
    set uploadObj = CreateObject("eWepAutoSvr.EkFile")
    ServerLocation = "/images"      ' Hard coded the location for this sample.

    strNewFileName = uploadObj.EkFileSave(BinaryFormData, "uploadfilephoto", _
        Server.MapPath(ServerLocation), ErrorCode, "makeunique")

    g_iFileCount = uploadObj.FileCount()
    If g_iFileCount > 0 then
        Do while iFileIdx < g_iFileCount
            iFileIdx = iFileIdx + 1
            Set fileObj = uploadObj.FileObject(iFileIdx)
            strNewFileName = fileObj.FileName()
            strFileLoc = "HTTP://" & Request.ServerVariables("SERVER_NAME") & ServerLocation &
"/" & strNewFileName
            fileObj.FileUrl(strFileLoc)
            fileObj.Thumbnail(CreateThumbnail(strFileLoc))
            fileObj.ThumbReference(ExtractThumbnailRef(strFileLoc))
        loop

        'Retrieve global data
        g_strDataIslandID = uploadObj.ResponseID()
        g_iClientMajorRev = uploadObj.ClientMajorRev()
        g_iClientMinorRev = uploadObj.ClientMinorRev()

        Response.Write(uploadObj.ResponseData())
    End If
End Sub
%>

<% If g_iFileCount > 0 Then %>

<% If 1 = g_iClientMajorRev then %>
<h3 style="align:center">Uploaded File Information</h3>
<table DATASRC="#<%Response.Write(g_strDataIslandID)%>" cellpadding=3 border=1>
<thead><tr><th>Thumbnail</th><th>Image Display</th><th>Description of Image</th></tr></thead>
<tr>
    <td><A DATAFLD="THUMBHREF" target="_blank"><IMG DATAFLD="THUMBURL"></a></td>
    <td><IMG DATAFLD="FURL"></div></td>
    <td><div DATAFLD="DESC"></div></td>
</tr>

```

```
</table>
<% End If %>

<% Else %>
<p>No files were uploaded.</p>
<% End If %>

</body>
</html>
```

The above code includes a table to illustrate how you can use XML data in the response page. It is *not* required with the XML data.

Steps to Receiving Content

Step 1 - Act on the Command

The `uploadcontent` command signals to the receiving server that a file is included in the posting. The command is retrieved from the `actiontyp` field in the posted form.

Step 2 - Extract the Content

Information about the uploaded content is in the "content_title", "content_size", "content_type", and "content_description" fields. The actual content is in the "content_text" field.

The received content can be in HTML, XML, or RTF format. The format received is determined by the client side scripting and configuration.

Below is an ASP line that extracts the content:

```
strContent = objUpload.EkFormFieldValue(binaryFormData, "content_text", ErrorCode)
```

Step 3 - Save the Content

The receiving script saves the content in the mechanism that it requires. Below is ASP saving the content to the database:

```
AddNewContentToDatabase SQLFilter(strTitle), SQLFilter(strContent)
```

Step 4 - Return a Response

The editor displays the response in the editor itself. Because of this, the client should generate a response that the user understands.

Below is an ASP example that generates a response which confirms the content upload to the user.

```
strResp = "<html><body>"
If "New" = strDesc Then
    strResp = strResp & "<H2>New Content Received</h2>"
    AddNewContentToDatabase strTitle, strHtml
Else
    strResp = strResp & "<H2>Updated Content Received</h2>"
    UpdateContentInDatabase strTitle, strHtml, strID
End If
```

```

strResp = strResp & "Content Title:&nbsp;&nbsp;" & strTitle & "<br>"
strResp = strResp & "<hr>body></html>"
Response.Write(strResp)

```

EWepAutoSvr Object API

These methods enable the file upload feature.

- [ClientMajorRev](#)
- [ClientMinorRev](#)
- [EkFileSave](#)
- [EkFileSave2](#)
- [EkFormFieldValue](#)
- [EkFileSize](#)
- [FileObject](#)
- [FileCount](#)
- [responseData](#)

ClientMajorRev

Description

Returns the client's major revision number. The client sends its version number in the `ekclientversion` submission field.

Example

```
iClientMajorRev = uploadObj.ClientMajorRev()
```

ClientMinorRev

Description

Returns the client's minor revision number. The client sends its version number in the `ekclientversion` submission field.

Example

```
iClientMinorRev = uploadObj.ClientMinorRev()
```

EkFileSave

Description

This method takes a given post stream and extracts the uploaded file from it. It then uses the parameters to determine how to save the file.

The method also extracts information about the file being uploaded. This information is saved in the File object (which you obtain using the `FileObject` method) and is reflected in the response XML produced with the `responseData` method. To retrieve the client version information contained in this stream, use the `ClientMajorRev` and `ClientMinorRef` methods.

See Also: "ClientMajorRev" on page 540; "ClientMinorRev" on page 540

This method is included to be compatible with existing EktronFileIO scripts, so that only minimal changes are needed to incorporate this module into existing routines.

WARNING!

This method is obsolete and should not be used in future implementations. It exists for compatibility purposes only. Replaced by EkFileSave2.

Parameters

Parameter	Type	Description
BinaryFormData	Variant(String)	The entire form data in binary form.
FormFieldName	Variant(String)	The name of the field used in the original form. This was a Form field defined as <code>type="file"</code> .
DestinationDir	Variant(String)	The fully qualified path (for example, <code>c:\inetpub\wwwroot\test</code>).
ErrorCode	Variant(Number)	A user supplied variable. This is set to 0 (zero) for successful execution. Otherwise, it is set to a server error code.
NameConflict	Variant(String)	Determines the behavior when the requested filename conflicts with an existing file. Choose "makeunique", "overwrite" or "error". "error" is the default.
AcceptType	Variant(String)	Determines which file types the upload accepts (for example, <code>image/gif</code> , <code>application/msword</code>). Not supported in this release.
FilePermissionSetting	Variant(String)	Not supported in this release.
FileAttributes	Variant(String)	Not supported in this release.
ReturnString	Variant(String)	If ErrorCode (see above) is 0 (zero), this contains the filename used to store the file, including the full path. If ErrorCode is <> 0, this contains a matching error string.

Example

```
ReturnString = EkFileSave ( "BinaryFormData", "FormFieldName", "DestinationDir", ErrorCode,
[ "NameConflict" ], [ "AcceptType" ], [ "FilePermissionSetting" ], [ "FileAttributes" ] )
```

EkFileSave2

Description

See Also: "EkFileSave" on page 540

Parameters

Parameter	Type	Description
BinaryFormData	Variant(String)	The entire form data in binary form.
FormFieldName	Variant(String)	The name of the field used in the original form. This was a Form field defined as <code>type="file"</code> .
DestinationDir	Variant(String)	The fully qualified path (for example, <code>c:\inetpub\wwwroot\test</code>).
ErrorCode	Variant(Number)	A user supplied variable. This is set to 0 (zero) for successful execution. Otherwise, it is set to a server error code.
NameConflict	Variant (String)	Determines the behavior when the requested filename conflicts with an existing file. Choose "makeunique", "overwrite" or "error". "error" is the default.
NewFilename	Variant (String)	If this value is present and is not an empty string, this filename is used by EkFileSave2 to write the file to the filesystem. This parameter lets the programmer override the forms filename to which the file data is attached.
AcceptType	Variant(String)	Determines which file types the upload accepts (for example, <code>image/gif</code> , <code>application/msword</code>). Not supported in this release.
FilePermissionSetting	Variant(String)	Not supported in this release.
FileAttributes	Variant(String)	Not supported in this release.
ReturnString	Variant(String)	If ErrorCode (see above) is 0 (zero), this parameter contains the filename used to store the file, including the full path. If ErrorCode is $<> 0$, this contains a matching error string.

Example

```
ReturnString = EkFileSave2 ("BinaryFormData", "FormFieldName", "DestinationDir", ErrorCode,
["NameConflict"], ["NewFilename"], ["AcceptType"], ["FilePermissionSetting"],
["FileAttributes"])
```

EkFormFieldValue

Description

This method retrieves the value of a specific field in the binary form data passed to it. It can retrieve the value of a text area, a list box selection, or any other item that exists in the file.

Parameters

Parameter	Type	Description
BinaryFormData	Variant(String)	The entire form data in binary form.
FormFieldName	Variant(String)	The name of the field used in the original form. Any form field name used in your original form. Fields with <code>type="file"</code> only return the filename submitted by the user.
ErrorCode	Variant(Number)	This is a user supplied variable. This is set to 0 (zero) for successful execution. Otherwise, it is set to a server error code.
ReturnedFormField Value	Variant(String)	If ErrorCode (see above) is 0 (zero), this contains the actual form field value. If ErrorCode is <> 0, this contains a matching error string.

Example

```
ReturnedFormFieldValue = fileObj.EkFormFieldValue("BinaryFormData", "FormFieldname", ErrorCode)
```

EkFileSize

Description

See Also: "["FileSize"](#) on page 547

Parameters

Parameter	Type	Description
BinaryFormData	Variant(String)	The entire form data in binary form.
FormFieldName	Variant(String)	The name of the field used in the original form. Only fields with <code>type="file"</code> return a valid size. The size is in bytes.
ErrorCode	Variant(Number)	This is a user-supplied variable. This is set to 0 (zero) for successful execution. Otherwise, it is set to a server error code.

Parameter	Type	Description
ReturnedSize	Variant(String or long)	If ErrorCode (see above) is 0 (zero), this field contains the form file size in bytes. If ErrorCode is <> 0, this field contains a matching error string.

Example

```
ReturnedSize = fileObj.EkFileSize("BinaryFormData", "FormFieldname", ErrorCode)
```

FileObject

Description

Returns the object related to the name returned from the file upload. This object is used to set each value for the file.

See Also: "[EkFileObject API](#)" on page 544

Parameters

Parameter	Description
FileName	Either the name of the file returned from EkFileSave or the 1-based index into the uploaded files.

Example

```
set fileObj = uploadObj.FileObject(strFileName)
```

FileCount

Description

Returns the number of files uploaded. If enumerating, use the indexes into the files with the FileObject method.

Example

```
iFileCount = uploadObj.FileCount()
```

responseData

Description

This returns the response data stream that should be sent back to the client side. The return value should be placed into the content returned.

Example

```
strResponse = uploadObj.ResponseData();
Response.Write(strResponse)
```

EKFileObject API

These methods are available to the client script through the file object.

- Description
- FileDimensions
- FileError
- FileID
- FileName
- FileSize
- FileType
- FileUrl
- Fragment
- Thumbnail
- ThumbReference

Description

Description

This sets the description given to the file. Description is used in the title and alt attributes in an image tag for images, and as the link text in other files.

Parameters

Parameter	Description
url	The full URL to the resulting file. This is the path that a browser uses to reference the file.

Example

```
fileObj.FileUrl("http://www.ektron.com/images/gif/me.gif")
```

FileDimensions

Description

Sets the dimensions of the image shown. If this is not called for, any value set to 0 uses the image's dimensions.

Parameters

Parameter	Description
width	The width to show the image. A value of 0 means to use the image's dimension.
height	The height to show the image. A value of 0 means to use the image's dimension.
border	The border width around the image.

Example

This puts a border around the image.

```
fileObj.FileDimensions(0, 0, 1)
```

FileError**Description**

This sets error values from the upload process. Normally, this is a server error.

If ekFileIO had an internal error, and this is not called by the client script, it places its internal error into these values.

Parameters

Parameter	Description
value	The value of the error. A zero (0) means no error. If there is an error internal to ekFileIO and 0 is set through this parameter, the internal error is used.
desc	The description of the error. A server may want to send a translated version of this string.

Example

```
fileObj.FileError(102, "This file is not allowed on the system.")
```

FileID**Description**

If the client wants to assign an ID value to the uploaded file for use in the content, use this method to specify the value.

This is not the ID used in the XML data to identify the file element group. Instead, the server side script assigns this ID as a value in the client content.

If the file is an image or a thumbnail is specified, this value is placed within an tag. If the file is not an image, and no thumbnail is specified, this value is placed within an <A> tag.

Parameters

Parameter	Description
id	The value to use as an ID.

Example

```
fileObj.FileID("img1027")
```

FileName

Description

This returns the resulting file name. It is equivalent to what is normally returned from the EkFileSave method.

It is a non-modifiable value.

Example

```
strNewFileName = fileObj.FileName()
```

FileSize

Description

This returns the size of the file in bytes.

Example

```
iSize = fileObj.FileSize()
```

FileType

Description

This specifies the file type. If this method is not called, EWepAutoSrv tries to determine file type from the file's extension or the file type sent by the client.

The server side script calls this method when processing changes to file type, or when the file type is not the expected file type. For example, a .BMP file is converted to a .GIF file, or a Word document is converted to a .PDF file.

If no thumbnail is given, this entry determines how the resulting file is represented in the content. The following table describes how different file types are handled when there is no thumbnail.

Type	How Handled if no Thumbnail
image/gif	The value set in FileUrl is placed within an tag. The value set with the Description method is placed within the title and alt attributes. There is no link created. The "gif" portion (shown here) is set to the specific type of image file. If not assigned otherwise, EWepAutoSrv sees these extensions as an image: "gif", "tif", "bmp", "tga", "emf", "wmf", "img", "jpg", "jpeg", "pic", "pcx", "png".
other (Default or a given unknown type)	The value set in FileUrl is placed in an <A> tag. The value set with the Description method is placed as text within the link. The extension is appended to the type as with image. <i>See Also:</i> "Description" on page 545

Since this is a text field set by the script, other types can be implemented in the future on the client side.

Parameters

Parameter	Description
type	The text type. These values are recognized: <ul style="list-style-type: none"> • image/gif • other

Example

```
fileObj.FileType( "image/gif" )
```

FileUrl

Description

This sets the full URL to the resulting file. It may be the file's name with a numbered extension or a completely different file type.

This must be the full reference location which includes a protocol (HTTP/HTTPS), server (www.yahoo.com), and full path. A relative path is not allowed.

NOTE	If the Thumbnail method is called with a value, its URL value is used as the image source value in the content.
-------------	---

Parameters

Parameter	Description
url	The full URL to the resulting file. A browser would use this path to reference the file.

Example

```
fileObj.FileUrl("http://www.ektron.com/images/gif/me.gif")
```

Fragment**Description**

If you want to determine how the resulting image appears in the content, specify the HTML using this method.

If you specify an HTML fragment, the client side performs no processing and offers the user no options to modify the content. The fragment goes into the content as given. The user must work through the HTML or XML functionality to modify the content.

NOTE	The example below is generally what the client side editor does with thumbnail content. The main difference is that the client implements a tag around both the thumbnail and the descriptive text. This allows the content to exist within any section, including a paragraph.
-------------	--

Parameters

Parameter	Description
url	The full URL to the resulting file. A browser would use this path to reference the file.

Example

```
fileObj.Fragment(<table><tr><td><img src='mythumbnail.gif'></td></tr><tr><td>Photo of my thumb</td></tr></table>")
```

Thumbnail**Description**

This sets a thumbnail file to use in place of the uploaded file. It could be a thumbnail generated from an image, or a thumbnail to use as an icon for an uploaded file.

A thumbnail always has a link attached to it. If the ThumbReference method is not called, the URL of the resulting file is used.

See Also: "[ThumbReference](#)" on page 550

Parameters

Parameter	Description
url	The URL of the thumbnail file

Example

```
fileObj.Thumbnail("http://www.ektron.com/images/thumbnails/me.gif");
```

ThumbReference

Description

Use this parameter if a thumbnail needs to refer to a file other than the one that is uploaded or if a thumbnail needs to call a page with parameters.

This must be the full reference location. It must include the protocol (HTTP/ HTTPS), the server (www.yahoo.com), and the full path. A relative path is not allowed. The value must also be encoded, so for example, any '&' characters must be entered as "&".

If this value is given, the value set with the FileUrl method is not used as a reference. If this value is not given or is empty, the FileUrl value is used as a reference.

Parameters

Parameter	Description
url	The URL of a file or a page with parameters. The string must be encoded.

Example

```
fileObj.ThumbReference("http://story.news.yahoo.com/news?tmpl=story&u=/021218/161/2w6dq.html");
```

XML Element Descriptions

This section describes these XML elements. You use them to build the return XML data. See Also: "Step 5 – Build the Return XML Data" on page 532.

- [DBORDER](#)
- [DESC](#)
- [DHEIGHT](#)
- [DWIDTH](#)
- [FERROR](#)
- [FID](#)
- [FILEINFO](#)
- [FRAGMENT](#)
- [FSIZE](#)
- [FSRC](#)
- [FTYPE](#)
- [FURL](#)
- [THUMBURL](#)
- [THUMBREF](#)
- [UPLOAD](#)

DBORDER

Description

The border to use around the image or thumbnail. If this value is not set or zero (0), no border appears.

Example

```
<DBORDER>1</DBORDER>
```

DESC

Description

This contains the file description. It is sent from the client, but is also returned since the server may want to change it.

If the file type is an image or a thumbnail is given, this value is used in the alt and title attributes of the tag.

If the file type is other than an image, this value is the text contained within the link (that is, <A>) tags.

Example

```
<DESC>'Rings' Leads Charge to Record</DESC>
```

DHEIGHT

Description

The display height to use for the image or thumbnail. If this value is not set or 0, the height of the image is used.

Example

```
<DHEIGHT>180</DHEIGHT>
```

DWIDTH

Description

The display width to use for the image or thumbnail. If this value is not set or 0, the width of the image is used.

Example

```
<DWIDTH>300</DWIDTH>
```

FERROR

Description

Contains any error from the uploading and processing of the file. It may state that there are no upload permissions, that the file does not meet a set of criteria, or that there was a technical issue.

Example

```
<FERROR val="0"></FERROR>
```

Attributes

Attribute	Description
val	An integer value representing the error. A value of 0 means no error. The default is 0.

FID

Description

If the client wants to assign an ID value to an uploaded file for later processing, it uses this element to specify its value.

This is not the ID used in the XML data to identify the file element group. It is an ID assigned by the server side script to have as a value in the client content.

If the file is an image, or a thumbnail is specified, this value is placed within the tags.

If the file is other than an image, and there is no thumbnail specified, this value is placed within the <A> tags.

Example

```
<FID>img1027</FID>
```

FILEINFO

Description

This element contains individual pieces of information about an uploaded file.

Example

```
<FILEINFO ID="0" discard="false">
```

Attributes

Attribute	Description
ID	Each file contained in the list of files must have a unique ID.
discard	A server may accept the file, process the data, and generate data, but the server may not want this file to be available as a reference in the content. Set this value to <code>true</code> to prevent the editor from offering the file as a reference.
width	The width to show the image or thumbnail. If set to 0, the image width is used. The default is 0.
height	The height to show the image or thumbnail. If set to 0, the image height is used. Default is 0.
border	The border width to apply to the image or thumbnail.
style	Style information to apply to the image, thumbnail, or link. If this is specified, width, height, and border attributes are ignored.

FRAGMENT

Description

If the server does not want to have the Fileinfo information formatted automatically, use this field to specify an HTML fragment to insert into the content at the current location. The fragment determines how the resulting image appears in the content.

If a fragment is specified, no image or link functionality is invoked on the returned data. The HTML fragment is just inserted at the current location.

NOTE

The example below is generally what the client side editor does with the given thumbnail content. The main difference is that the client implements a `` tag around both the thumbnail and the descriptive text. The fragment allows the content to exist within any section, including a paragraph.

Example

```
<FRAGMENT>
<table>
<tr>
<td></td>
</tr>
<tr>
<td>Photo of my thumb</td>
</tr>
</table>
</FRAGMENT>
```

FSIZE

Description

The size in bytes of the uploaded file.

Example

```
<FSIZE>107342681</FSIZE>
```

FSRC

Description

The original name of the source file. Ektron recommends using only the name and not the full path sent by the client.

This is *not* the modified name. (The modified name goes into the [FURL](#) element.)
This is the name as given by the client.

The client script cannot affect this through the EWepAutoSvr interface. The module sets this value internally from the upload.

Example

```
<FSRC>iraqshow.jpg</FSRC>
```

FTYPE

Description

The resulting file type. A BMP file might be converted to a GIF file, or a Word document converted to a PDF file.

If no thumbnail is provided, this entry determines how the resulting file is represented in the content. The following table describes how file types are handled when no thumbnail is provided.

Type	How Handled
image/gif	<p>The FURL value is placed within tags. The DESC value is placed within the title and alt attributes. There is no link created.</p> <p>The gif portion (shown here) is set to the specific type of image file.</p>

Type	How Handled
other	(Default or a given unknown type) The FURL value is placed within <A> tags. The DESC value is placed as text within the link.

Example

```
<FTYPE>image/jpeg</FTYPE>
```

FURL**Description**

The full URL to the resulting file. It may be the file's name with a numbered extension or a completely different file type.

This must be the full reference location. That is, it must include the protocol (HTTP/HTTPS), server (www.yahoo.com), and full path. A relative path is not allowed.

NOTE

If there is a THUMBURL element, its URL value is used as the image source value.

Example

```
<FURL>http://us.news2.yimg.com/us.yimg.com/p/nm/20021218/mdf171290.jpg</FURL>
```

THUMBNAIL**Description**

A thumbnail may be assigned to the uploaded file. If the file is a Word document, the server may want to assign an icon to the uploaded document rather than a text link.

In the content, the thumbnail represents the uploaded file. The thumbnail contains a link that the user can click to access the uploaded file.

```
<a href="http://story.news.yahoo.com/news?tmpl=story&u=/021218/161/2w6dq.html">

</a>
```

When a thumbnail is specified, it is used in place of the FURL value returned. Normally, a reference URL is returned with a thumbnail. But, if no reference is sent, the URL of the resulting file is returned.

Example

```
<THUMBNAIL>http://us.news1.yimg.com/us.yimg.com/p/nm/20021218/amdf171290.jpg</THUMBNAIL>
```

THUMBHREF**Description**

Use this element if a thumbnail needs to refer to a file other than the one that is uploaded, or needs to call a page with parameters.

This must be the full reference location. It must include the protocol (HTTP/HTTPS), the server (www.yahoo.com), and the full path. A relative path is not allowed. The value must also be encoded, so any ampersand (&) characters must be given as &:.

If this value is given, the value of FURL is not used as a reference.

If this value is not given or it is empty, the FURL value is used as a reference.

See Also: "THUMBNAIL" on page 555, "FURL" on page 555

Example

```
<THUMBREF>http://story.news.yahoo.com/news?tmpl=story&u=/021218/161/2w6dq.html
</THUMBREF>
```

UPLOAD

Description

The root element of the data island. It contains all information about and the results of the upload.

Image Upload Response Example with Proprietary Information

You can include any information with the image receipt XML information. The response is considered valid as long as the data island is defined. The extra information returned by the server can be processed by the client side scripting.

Below is an example response that contains HTML tags. When the editor receives this, it does not display the HTML, but does find and parse the XML information. It also contains an extra tag, ServerInfo, which returns information to the client.

```
<html>
<head>
    <title>Posted File Received</title>
</head>

<body>
<h1>File Recieved</h1>
<p>The file posted is stored in the central repository for selection.
The content is modified to reflect its location.</p>
<br>
<p>Thank you.</p>

<XML ID=EktronFileIO>
<?xml version="1.0"?>
<UPLOAD>
    <ServerInfo>Image Stored In User Group</ServerInfo>
    <FILEINFO ID="0" discard="False">
        <FSRC>C:\Inetpub\wwwroot\Arrows\next0.gif</FSRC>
        <FURL>http://www.echo.com/ewebeditpro3/upload/me(1).gif</FURL>
        <FID></FID>
        <FSIZE>128</FSIZE>
        <DESC></DESC>
        <THUMBURL></THUMBURL>
        <THUMBREF></THUMBREF>
```

```

<FTYPE>image/gif</FTYPE>
<DWIDTH>0</DWIDTH>
<DHEIGHT>0</DHEIGHT>
<DBORDER>0</DBORDER>
<FRAGMENT></FRAGMENT>
<FERROR value="0"></FERROR>
</FILEINFO>
</UPLOAD>
</XML>

</body>
</html>

```

ColdFusion Example

Here is an example of how ColdFusion gathers the file and assembles the data.
All of the work is done within the script as defined in the document.

```

<cfinclude template="ewebeditprodefinedsn2.cfm">

<cfinclude template="#trim(replace(form.editor_media_path, "-", "", "ALL"))#/imagepath.cfm">

<cfset variable.uploadcommand="#form.actiontyp#">
<cfset variable.ErrorNumber="0">
<cfset variable.ErrorDesc="">

<cfif variable.uploadcommand eq "uploadcontent">
    <cfset variable.contenttitle="#form.content_title#">
    <cfset variable.contentsize="#form.content_size#">
    <cfset variable.contentdesc="#form.content_description#">
    <html>
        <body>
            <H1>Content Received</h1>
            <p style='color:red'>The receiving page <i>does not</i> save the posted content on the
server. <b>Content is not saved.</b></p>
            <p style='color:red'>Click on 'Undo' to restore the previous content.</p>
            <br>
            Content Title:&nbsp;&nbsp;
            <cfoutput>#variable.contenttitle#</cfoutput>
            <br>
            Content Size:&nbsp;&nbsp;
            <cfoutput>#variable.contentsize#</cfoutput>
            <br>
            Content Description:&nbsp;&nbsp;
            <cfoutput>#variable.contentdesc#</cfoutput>
            <br>

        </body>
    </html>
</cfif>

<cfif variable.uploadcommand eq "uploadfile">
    <html>
        <body>
            <H1>File Received</h1>
            <p>Uploading the file to the server.</p>

```

```

<CF_ewebeditproupploadfile
    allowexts="#trim(replace(form.extensions, "-", "", "ALL"))#"
    destdir="#variable.DestDir#"
    renamefile="Yes"
    uploadfile="#form.uploadfilephoto#"
    nameconflict="MAKEUNIQUE"
    TempDir="#variable.DestDir#">
<cfif errorlevel>
    <p>Error saving the file on the Cold Fusion server. Error level is #errorlevel#. </p>
    <cfset variable.ErrorNumber="1">
    <cfset variable.ErrorDesc="Error saving the file on the Cold Fusion server.">
<cfelse>

    <CFQUERY NAME="i_media" DATASOURCE="#DSN#">
        INSERT INTOmedia_tbl (media_title, media_path, media_filename, media_upload_date,
        media_filesize, user_name,
        site_id, media_deleted, extension_id, media_width, media_height)
        VALUES('#uploadedfilename#', '#trim(replace(form.editor_media_path, "-", "", "ALL"))#/',
        '#uploadedfilename#', #DateFormat(Now(), "MM/DD/YY")#,
        #trim(replace(form.file_size, "-", "", "ALL"))#, 'user name',
        0, 0, 1, 0, 0)
    </CFQUERY>

</cfif>

<XML ID="EktronFileIO">
<?xml version="1.0"?>
<UPLOAD>
    <FILEINFO ID="0" discard="False">
        <FSRC><cfoutput>#trim(original_name)#</cfoutput></FSRC><!-- Original source given by
the client should go in here --->
        <FURL><cfoutput>#trim(replace(form.editor_media_path, "-", "", "ALL"))#/
#uploadedfilename#</cfoutput></FURL>
        <FID></FID>
        <FSIZE>342</FSIZE>
        <DESC>My Description</DESC>
        <THUMBURL></THUMBURL>
        <THUMBHREF></THUMBHREF>
        <FTYPE>image/gif</FTYPE>
        <DWIDTH></DWIDTH>
        <DHEIGHT></DHEIGHT>
        <!--
        Note:
            ColdFusion chokes on DBORDER because ColdFusion tags used to start with DB
            so it automatically converts DB to CF which becomes CFORDER which is an invalid tag
        -->
        <cfoutput><DB</cfoutput>ORDER></DB<cfoutput>ORDER></cfoutput>
        <FRAGMENT></FRAGMENT>
        <cfif variable.ErrorNumber eq "0">
            <FERROR value="0"></FERROR>
        <cfelse>
            <FERROR value="1"><cfoutput>#variable.ErrorDesc#</cfoutput></FERROR>
        </cfif>
        </FILEINFO>
    </UPLOAD>
</XML>

```

```

</body>
</html>

</cfif>

<cfif variable.uploadcommand neq "uploadcontent">
  <cfif variable.uploadcommand neq "uploadfile">

    <html>
    <body>
    <H1>Content Received</h1>
    <p>Upload command is not recognized.</p>
    <p>It was [ <cfoutput>#variable.uploadcommand#</cfoutput> ]</p>

    </body>
    </html>

  </cfif>
</cfif>

```

ASP Example

Here is an example of how ASP gathers the file and assembles the data. Most of the work of creating the XML is within the eWepAutoSvr.dll module.

```

<!-- #include file="functions.asp" -->
<%
' mediaautoreceive.asp
' Receives files without involving the ASP database user interface.
'
' The functions.asp script holds the database functionality.
%>

<%
  Dim g_LogicalRefDestination
  Dim g_objUpload
  Dim g_binaryFormData

  Set g_objUpload = CreateObject("eWepAutoSvr.EkFile")
  g_binaryFormData = Request.BinaryRead(Request.TotalBytes)

  ' Recieve and save the files
  ProcessSubmittedForm

  .....
  ' Examines the submitted for to determine what
  ' the client is uploading and to perform the
  ' appropriate operation.
Sub ProcessSubmittedForm()
  Dim strCommand, ErrorCode

  ' Extract the "actiontyp" field.
  ' This contains the upload command.

```

```

strCommand = g_objUpload.EkFormFieldValue(g_binaryFormData, "actiontyp", ErrorCode)

' These are the possible commands:
If strCommand = "uploadfile" Then
    ReceiveSubmittedFiles ' Saves the submitted files.
ElseIf strCommand = "uploadcontent" Then
    ReceiveContent
Else
    Response.Write("<html><body><h1>Unknown Posting.</h1></body></html>")
End If
End Sub

-----
' This function will receive the files and send back
' the required response data. There is no processing
' of the files and there is no affecting the file data.
Sub ReceiveSubmittedFiles()
    Dim objFile, iErrorCode
    Dim strLogicalRefDest, strFileAltTitle, strReqWebRoot, strImageDate
    Dim iFileSize, iExtensionID, iWidth, iHeight, strFileType

    strLogicalRefDest = g_objUpload.EkFormFieldValue(g_binaryFormData, "editor_media_path",
iErrorCode)
    strFileAltTitle = g_objUpload.EkFormFieldValue(g_binaryFormData, "file_title", iErrorCode)
    strReqWebRoot = g_objUpload.EkFormFieldValue(g_binaryFormData, "web_media_path",
iErrorCode)
    strImageDate = g_objUpload.EkFormFieldValue(g_binaryFormData, "img_date", iErrorCode)
    iFileSize = g_objUpload.EkFormFieldValue(g_binaryFormData, "file_size", iErrorCode)
    iExtensionID = g_objUpload.EkFormFieldValue(g_binaryFormData, "extension_id", iErrorCode)
    iWidth = g_objUpload.EkFormFieldValue(g_binaryFormData, "width", iErrorCode)
    iHeight = g_objUpload.EkFormFieldValue(g_binaryFormData, "height", iErrorCode)
    strFileType = g_objUpload.EkFormFieldValue(g_binaryFormData, "file_type", iErrorCode)

    strNewFileName = g_objUpload.EkFileSave(g_binaryFormData, "uploadfilephoto", _
        Server.MapPath(strLogicalRefDest), iErrorCode, "makeunique")

    If g_objUpload.FileCount() > 0 then
        Set objFile = g_objUpload.FileObject(1)
        strNewFileName = objFile.FileName()
        objFile.FileUrl(MakeMediaPathName(strReqWebRoot, strNewFileName)) ' see:
functions.asp

        AddFileToDatabase strFileAltTitle, strReqWebRoot, strNewFileName, strImageDate,
iFileSize, iExtensionID, iWidth, iHeight

        Set objFile = Nothing
    End If

    Response.Write(g_objUpload.ResponseData())
End Sub

-----
' This routine processes the submission of the
' content contained within the eWebEditPro editor.
Sub ReceiveContent()

```

```

Dim strResp
Dim ErrorCode
Dim strTitle
Dim strHtml
Dim strID

strTitle = SQLFilter(g_objUpload.EkFormFieldValue(g_binaryFormData, "content_title",
ErrorCode))
strHtml = SQLFilter(g_objUpload.EkFormFieldValue(g_binaryFormData, "content_text",
ErrorCode))
strID = g_objUpload.EkFormFieldValue(g_binaryFormData, "content_description", ErrorCode)

strResp = "<html><body>

If "New" = strID Then
    strResp = strResp & "<H2>New Content Received</h2>
    AddNewContentToDatabase strTitle, strHtml
Else
    strResp = strResp & "<H2>Updated Content Received</h2>
    UpdateContentInDatabase strTitle, strHtml, strID
End If

strResp = strResp & "Content Title:&nbsp;&nbsp;" & strTitle & "<br>
' strResp = strResp & "Content Size:&nbsp;&nbsp;" &
g_objUpload.EkFormFieldValue(g_binaryFormData, "content_size", ErrorCode) & "<br>
' strResp = strResp & "Content Description:&nbsp;&nbsp;" & strID & "<br>

' strResp = strResp & "Content Type:&nbsp;&nbsp;" &
g_objUpload.EkFormFieldValue(g_binaryFormData, "content_type", ErrorCode)

' strResp = strResp & "<br>
' strResp = strResp & "<H3>Submitted Content Below</h3><hr>
' strResp = strResp & Server.HTMLEncode(strHtml)

strResp = strResp & "<hr>

strResp = strResp & "</body></html>

Response.Write(strResp)
End Sub

%>
```

Automatic Upload Object

You can programmatically control the Automatic Upload feature through a Object Interface, available through the Automatic Upload Object Interface.

For example:

```
objMedia = objEditor.MediaFile();
objAutoUpload = objMedia.AutomaticUpload();
objAutoUpload.AddFileForUpload(strFileName, strDescription);
```

See Also: "[Media File Object](#)" on page 20

Media File Object Properties

The Automatic Upload Object Interface supports the standard way of setting and retrieving property values, such as `setProperty`, `getProperty`, and `getPropertyString`.

See Also: "[Method: getProperty](#)" on page 90, "[Method: getPropertyString](#)" on page 91, "[Method: setProperty](#)" on page 124

The Media File Object has a few unique properties and several other properties that are a subset of the media object properties.

See Also: "[Property: TransferMethod](#)" on page 157; "[Property: ServerName](#)" on page 146

Automatic Upload Object Properties as a Subset of the Media Object Settings

The definitions for the following automatic upload properties are almost identical to the larger media object properties. They differ because they affect only the automatic upload mechanism, having no effect on the larger media object settings.

- ["Property: LoginName" on page 146](#)
- ["Property: LoginRequired" on page 147](#)
- ["Property: Password" on page 147](#)
- ["Property: TransferRoot" on page 147](#)
- ["Property: ValidExtensions" on page 147](#)
- ["Property: WebRoot" on page 147](#)

NOTE

To set the server-side receiving script, use the `TransferMethod` property. See Also: "[Property: TransferMethod](#)" on page 157

Content Upload

The content upload feature lets a user upload content to the server. The server returns a response in the editor. For instance, the response could summarize the content that was uploaded to the server.

An example of this feature would be a nurse who needs patient information. The nurse enters patient data, uploads it, and receives information back from the server about the patient without refreshing the page. The nurse could then correct information on the received data and submit it to the server.

Content Upload, part of the Automatic Upload feature, works like Automatic Upload in that

1. Content is uploaded in a form.
2. The server retrieves the field value.
3. The server responds to the client.

NOTE

The Content Upload feature is configured in the configuration data. So, unless you need to change something, the client scripting does not need to change the configuration for a client upload.

Retrieving Content from eWebEditPro+XML

To retrieve editor content, you can use the upload command or the GetContent method. Both use a standard set of content types to specify the kind of information to retrieve from the editor. As examples

- client side JavaScript uses GetContent to retrieve the HTML header for processing
- the content upload command sends the content as RTF to the server

The rest of this chapter explains how to use the content upload feature through these subtopics:

- "The Content Upload Command" on page 563
- "The Receiving Page" on page 568
- "Content Types" on page 570
- "Content Setting API" on page 564

The Content Upload Command

The Content Upload's command is cmdmfuuploadcontent. When this command is given to the editor (either through the menu bar or client scripting), the content is uploaded to the server. This content does not reach the client-side JavaScript or the form.

To enable the Content Upload feature in the user interface, add the `cmdmfuuploadcontent` command button in the `interface` section of the configuration data to a toolbar. If the button needs to be added from the client script, use the Toolbar object interface to add it. Although you can configure the upload using the Automatic Upload Object Interface, only the string command can execute the upload. (See Also: "Automatic Upload Object Interface Properties" on page 564)

The command's arguments are listed below.

Argument	Description
String Param	The requested content type to post to the server. See Also: "Content Type Categories" on page 570
Long Param	Not used

Content Setting API

Two API methods can be used to retrieve content from the editor and set content back to the editor.

- "Method: GetContent" on page 85
- "Method: SetContent" on page 119

You can use the methods with client side JavaScript to extract or set information about the content. The JavaScript can place the extracted information in a field and post it to the server or process it on the client side.

Automatic Upload Object Interface Properties

The Automatic Upload Object Interface has the following properties, which let you configure the upload to the server.

Property Name	Type	Description
ContentTitle	String	The title of the content being uploaded. The title is set externally to the editor, and can be set within the ready notification. The server receives this value when the content is posted.
ContentDescription	String	A description of the content. The description is set externally to the editor, and can be set within the ready notification. The server receives this value when the content is posted.

Property Name	Type	Description
GetContentType	String	<p>Specifies the type of content to post to the server when content is uploaded through the internal Content Upload mechanism. To see a list of valid values, go to "Content Type Categories" on page 570.</p> <p><i>See Also:</i> "How Content Type is Determined" on page 576</p>

JavaScript Example

Below is a JavaScript example of using the Automatic Upload Object Interface.

```
function UploadEditorContent(sEditorName, sTitle, sDescription)
{
    var objAutoUpload;

    objAutoUpload = eWebEditPro.instances[sEditorName].editor.MediaFile().AutomaticUpload();

    objAutoUpload.ContentTitle = sTitle;
    objAutoUpload.ContentDescription = sDescription;

    eWebEditPro.instances[sEditorName].editor.ExecCommand( "cmdmfuuploadcontent", _
        "whole", 0);
}
```

Fields in the Posted Form

This section describes the fields used in the posted content upload form. The server receives the form when the content is uploaded.

Field	Description
actiontyp	The command of what posting this is. (Notice the missing 'e' in the name.) For content upload, the value is <code>uploadcontent</code> .
content_description	The description of the content. A content management site could put information (such as an ID) about the uploaded document into this field and then parse the information.
content_size	The size of the content in characters
content_text	The actual DHTML or XML content posted to the server.

Field	Description
content_title	The title given to the content. This is usually done by the user, but not restricted to this.
content_type	The type of content send. See " Content Types " on page 570 for a list of types.
ekclientname	The name of the client application. This normally has the "ekmediatransfer" value. Error check this.
ekclientversion	The version of the upload module on the client side. The version <i>must</i> be 1.0 or higher to support this upload mechanism. Error check this.
_isChanged	<p>A standard HTML hidden input field whose name is formed by concatenating the editor instance name with <code>_isChanged</code>. For example, if the editor name is <code>MyContent1</code>, the field name is <code>MyContent1_isChanged</code>.</p> <p>The value of the field is 0 if the content was not saved, and 1 if it was saved. For example, if the <code>eWebEditPro.instances[n].save()</code> method is called, the value is 1. Otherwise, the value of the field is 0.</p> <p>You can use this field in a server-side script to determine if a content field has changed. For multiple content fields with 'GetType' assigned, you can use this field to determine if the field values are valid.</p> <p>Example (ASP)</p> <p>For an editor defined as:</p> <pre><% =eWebEditProEditor("TextHTML1", "100%", 250, strContent1) %> <% =eWebEditProField("TextHTML1", "TextHTML1", "htmlbody", "", "") %> <% =eWebEditProField("TextHTML1", "TextOnly1", "", "text", "") %></pre> <p>The following script reads the values when the page is submitted. The 'TextOnly' field is only valid if the 'TextHTML1_isChanged' field does not contain the value "0".</p> <pre><% =Request.Form("TextHTML1") %> <% If Request.Form("TextHTML1_isChanged") <> 0 Then %>
<hr> <% =Request.Form("TextOnly1") %> <% End If %></pre>

Below is an ASP example of how to use these fields. The example receives content and returns it to the client as encoded HTML.

```
.....  

' This ASP routine processes the submission of the  

' content contained within the eWebEditPro editor.  

Sub ReceiveContent()  

    Dim strResp  

    Dim ErrorCode  

    strResp = "<html><body>"  

    strResp = strResp & "<H2>Content Successfully Received</h2>"  

    strResp = strResp & "<p style='color:red'>However, the sample page that received the  

content <i>does not</i> save the posted content on the server.</p>"  

    strResp = strResp & "<p style='color:red; font:bold'>The content is not saved.</p>"  

    strResp = strResp & "<p style='color:red'>Modify the sample receiving page to save the  

content or specify another receiving page that does save the content.</p>"  

    strResp = strResp & "<p style='color:red; font:bold'>Click on 'Undo' to restore your  

content.</p>"  

    strResp = strResp & "<br>"  

    strResp = strResp & "Content Title:&nbsp;&nbsp;" &  

g_objUpload.EkFormFieldValue(g_binaryFormData, "content_title", ErrorCode) & "<br>"  

    strResp = strResp & "Content Size:&nbsp;&nbsp;" &  

g_objUpload.EkFormFieldValue(g_binaryFormData, "content_size", ErrorCode) & "<br>"  

    strResp = strResp & "Content Description:&nbsp;&nbsp;" &  

g_objUpload.EkFormFieldValue(g_binaryFormData, "content_description", ErrorCode) & "<br>"  

    strResp = strResp & "Content Type:&nbsp;&nbsp;" &  

g_objUpload.EkFormFieldValue(g_binaryFormData, "content_type", ErrorCode)  

    strResp = strResp & "<br>"  

    strResp = strResp & "<H3>Submitted Content Below</h3><hr>"  

    strResp = strResp & Server.HTMLEncode(g_objUpload.EkFormFieldValue(g_binaryFormData,  

"content_text", ErrorCode))  

    strResp = strResp & "<hr>"  

    strResp = strResp & "</body></html>"  

    Response.Write(strResp)
End Sub
```

Steps to Receiving Content

Step 1 - Act on the Command

The `uploadcontent` command signals the receiving server that the posting includes a file. The command is retrieved from the `actiontyp` field of the posted form.

Step 2 - Extract the Content

The information about the uploaded content is contained within the "content_title", "content_size", "content_type", and "content_description" fields. The actual content is contained in the "content_text" field.

The content received can be in HTML, XML, or RTF format. The format received is determined by the client side scripting and configuration.

Below is an ASP line that extracts the content.

```
strContent = objUpload.EkFormFieldValue(binaryFormData, "content_text", ErrorCode)
```

Step 3 - Save the Content

The receiving script saves the content in the mechanism that it requires. Below is ASP code that saves the content to the database.

```
AddNewContentToDatabase SQLFilter(strTitle), SQLFilter(strContent)
```

Step 4 - Return a Response

Because **eWebEditPro+XML** displays the response in the editor, the client should generate a response that the user understands.

Below is an ASP example showing how to generate a response that confirms the content upload.

```
strResp = "<html><body>"  
If "New" = strDesc Then  
    strResp = strResp & "<H2>New Content Received</h2>"  
    AddNewContentToDatabase strTitle, strHtml  
Else  
    strResp = strResp & "<H2>Updated Content Received</h2>"  
    UpdateContentInDatabase strTitle, strHtml, strID  
End If  
strResp = strResp & "Content Title:&nbsp;&nbsp;" & strTitle & "<br>"  
strResp = strResp & "<hr></body></html>"  
Response.Write(strResp)
```

The Receiving Page

Like Automatic Upload, Content Upload uses a receiving page on the server. The form with the data is posted to the receiving page.

The receiving script determines how and where content should be saved. The content is usually stored as a string in a database.

If you use the receiving page specified for Automatic Upload, the content upload can occur on the client side with just the command. The server side administrator or CMS builder must create the receiving page.

That page is specified in the configuration data or in the Automatic Upload Object Interface. The code below illustrates where the Automatic Upload page is specified in the configuration data.

```
<transport ... >  
<autoupload type="[eWebEditProPath]/ewepreceive.asp" ... />
```

The code below illustrates the object interface API that sets the receiving page.

```
objAutoUpload.TransferMethod = "[eWebEditProPath]/ewepreceive.asp";
```

A command, uploadcontent, is sent to the page in the actiontyp field that indicates the purpose of the upload.

Creating a Receiving Page

The page receiving the content must follow these steps. (The steps match the Automatic Upload feature's rules for receiving images.)

NOTE

Although the examples provided use ASP, they could also use Cold Fusion, JSP, or any other server scripting that allows access to posted forms.

1. The receiving page looks for the command specified in the actiontyp field. Below is an ASP example.

```
' Examines the submitted for to determine what
' the client is uploading and to perform the
' appropriate operation.
Sub ProcessSubmittedForm()
    Dim strCommand, ErrorCode

    ' Extract the "actiontyp" field.
    ' This contains the upload command.
    strCommand = g_objUpload.EkFormFieldValue(g_binaryFormData, "actiontyp", _
        ErrorCode)

    ' These are the possible commands:
    If strCommand = "uploadfile" Then
        ReceiveSubmittedFiles  ' Saves the submitted files.
    ElseIf strCommand = "uploadcontent" Then
        ReceivePostedContent
    Else
        Response.Write("<html><body><h1>Unknown Posting.</h1></body></html>")
    End If
End Sub
```

2. Retrieve content from the content_text field.

```
StrContent = g_objUpload.EkFormFieldValue(g_binaryFormData, "content_text", _
    ErrorCode)
```

3. Retrieve additional information about the posted content from the content_title, content_size, content_description, and content_type fields.

```
strVal = g_objUpload.EkFormFieldValue(g_binaryFormData, "content_title", _
    ErrorCode)
strVal = g_objUpload.EkFormFieldValue(g_binaryFormData, "content_size", ErrorCode)
strVal = g_objUpload.EkFormFieldValue(g_binaryFormData, "content_description", _
    ErrorCode)
```

4. Store the information in a database. The page can also provide user feedback.

```
strResp = "<html><body>"  
strResp = strResp & "<H2>Content Received</h2>"  
strResp = strResp & g_objUpload.EkFormFieldValue(g_binaryFormData,  
"content_title", ErrorCode)  
strResp = strResp & "</body></html>"  
Response.Write(strResp)
```

Content Types

This section describes the supported content types and their limitations. They can be used as parameters of the GetContent method and the cmdmfuuploadcontent command.

What Happens if a Content Type is Not Supported

When *retrieving* content (that is, using the GetContent method), if a content type is not supported (either because the request is invalid or the content type is not supported under the editor's current mode), no string is returned.

When *setting* content, if a specified content type is not the content type being set, the resulting display in the editor is undefined.

Content Type Categories

Content types can be divided into these categories:

- Data Designer (see "[Supporting the Data Designer](#)" on page 601)
 - design information (only available with eWebEditPro+XML)
 - data collected from user (only available with eWebEditPro+XML)
- HTML
- XML (only available with eWebEditPro+XML)
- Text only
- Content in RTF format

The following tables describe the content types in each category.

Data Designer: Design Information

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
designbody	<p>The body of the design, which includes the defined data items and the default values.</p> <p>Before the data can be designed, all associated header information must be loaded.</p> <p>See "designheader" on page 571, "designdefinitions" on page 571, and "designformat" on page 571.</p>	not used	yes	no
designdataformat	<p>The transformation information, which is used to format the data entered by the user so that it can be displayed on a web page. (This is not returned with any other type.)</p>	not used	yes	no
designdefinitions	<p>The information required to validate the data entered by the user. This is the schema.</p> <p>The information is a piece of the data within the <i>designheader</i> type.</p>	not used	yes	no
designformat	<p>The information to transform the data design for use in data entry (that is, transformation information).</p> <p>The information is a part of the header within the <i>designheader</i> type.</p>	not used	yes	no
<u>designheader</u> <u>Note: Not implemented in this release.</u>	<p>All of the information needed to display data for data entry. This includes the schema and the transformation. It does not contain the data entered.</p> <p>See Also: "dataheader" on page 573.</p>	not used	no	no

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
designwhole	<p>The entire Data Design packet, including the schema, transformation, and default data.</p> <p>This packet is a complete set that can be stored and later set into the editor without needing any other types.</p>	not used	yes	yes
designpage	<p>An HTML representation of the page that contains the formatting around the data.</p> <p>This is <i>not</i> the transformation. It is the data that is edited during the design process. The schema and transformations are based on this source.</p>	not used	yes	yes

Data Designer: Data Collected From User

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
databody	<p>The body content, that is, the data items containing data entered by the user.</p> <p>Before the data can be displayed to the user for data entry, all associated header information must be loaded.</p> <p>See also "dataheader" on page 573, "datadefinitions" on page 572, and "dataformat" on page 573.</p>	Data Design to use with the data.	yes	no
datadefinitions	<p>The information required to validate information entered by the user. This is the schema.</p> <p>The information is a piece of the data in the <i>dataheader</i> type.</p>	not used	yes	no

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
datadesignpackage	<p>Data design mode: set or get the entire package of information</p> <p>Data entry mode: set the entire package of information</p> <hr/> <p>Note: Not recommended because the package is typically very large and can exceed the 64K limit.</p> <hr/>	not used	yes	yes
datadocumentxml	<p>Data design mode: get the default xml document</p> <p>Data entry mode: set or get the xml document</p>	not used	yes	yes
dataentryxslt	<p>Data design mode: get the XSLT for data entry mode</p> <p>Data entry mode: set the XSLT for data entry mode</p>	not used	yes	yes
dataformat	<p>The information to transform the data design for use in data entry.</p> <p>The information is a part of the header in the dataheader type.</p>	not used	yes	no
dataheader	<p>All information needed to display data for data entry. This includes the schema and transformation.</p> <p>This content type does not contain the data entered. See Also: "designheader" on page 571</p>	not used	no	no
dataindex	<p>An XML document's index information.</p> <p>See Also: "Indexing the Fields of a Data Design Document" on page 613</p>	not used	yes	no

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
datapresentation	The input data, with surrounding formatting, represented in HTML.	not used	no	no
datapresentationxslt	In Data Design mode: get the XSLT for the default presentation.		yes	no
datapresentationpage	<p>The default presentation HTML. This is created by transforming the xml data document with the presentation xslt.</p> <p>It is available in Design mode. Also available in Data Entry mode if the 'datapresentationxslt' was set.</p> <p>The datapresentationpage data may be saved by the host application (for example, a CMS) to avoid running a transform on the server.</p> <p>Note that while the datapresentationpage is HTML, it is not a whole HTML document. As a result, you can include it in a larger HTML document.</p>	not used	yes	no
dataschema	<p>Data design mode: get the XML schema that was designed</p> <p>Data entry mode: set the XML schema for the document</p>	not used	yes	yes
datawhole	The whole data packet, which contains the data design information and the data entered.	The simple data packet to use with the design.	yes	yes

HTML Information

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
htmlbody	The body of the content. Mainly valid in WYSIWYG or source view modes.	not used	yes	yes
htmlheader	The HTML header information.	not used	yes	yes
htmlwhole	The entire content. If the mode is WYSIWYG or source view, the full HTML is set or returned, and the cleaning level specified in the configuration is applied. If the mode is data design or data entry, the entire design/entry packet is set or returned.	not used	yes	yes

XML Information

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
xmlbody	The root tag and all contents within it.	The temporary transformation to use when loading the XML.	yes	yes
xmlheader	The header of the XML document. This is important for namespaces and other definitions.	not used	yes	yes
xmlwhole	The entire XML document.	The temporary transformation to use when loading the XML.	yes	yes

Plain Text

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
text	The text of the content. Formatting information not included.	not used	yes	no

Content in RTF Format

Content Type	Description	Parameters	Works with Get Content?	Works with Set Content?
rtf	The full content in RTF format.	not used	yes	yes

How Content Type is Determined

During a content upload, the following series of checks determines the content type being retrieved.

1. Is content type specified in the GetContent method? (See "[Method: GetContent](#)" on page 85.) If not, go to step 2.
2. Is it specified in the Automatic Upload Interface Object? (See "[Automatic Upload Object Interface Properties](#)" on page 564.) If not, go to step 3.
3. Is it specified in the configuration, specifically the mode attribute of standard element? (see "[The Mode Attribute](#)" on page 576.) If not, go to step 4.
4. htmlbody type is used.

The Mode Attribute

The mode attribute specifies which content type to publish when the type is not specified elsewhere. The attribute determines both what is returned from this API and what is posted with the cmdmfuuploadmedia string command.

Element Name: Standard

Attribute Name: mode

Type: String

Enumerated Values: See "[Content Type Categories](#)" on page 570

WeblImageFX

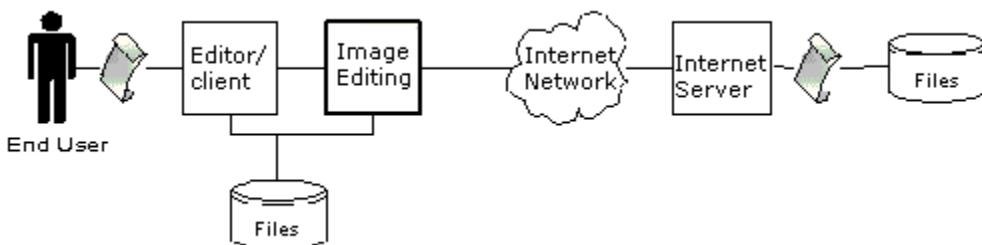
WeblImageFX is an external, add-on product available from Ektron.

WeblImageFX allows the user to

1. Load an image in an image editor
2. Modify the image in several ways, including
 - adjusting brightness, contrast, sharpness
 - adding text
 - changing its dimensions

3. Update the editor content with the new version of the image

The following diagram describes where WeblImageFX fits in with the other editor components.



The feature is installed to the `webroot\ewebeditpro5` directory by default. When the feature is installed on a client, the Webmaster uses the WeblImageFX object to control the feature's operation. The support of this feature involves

- additions to **eWebEditPro+XML**'s API
- additions to the configuration data
- an object available to the client

This section covers these topics.

- [Using the WeblImageFX Object](#)
- [Adding a Toolbar Button to Launch WeblImageFX](#)
- [New Configuration Variable](#)
- [WeblImageFX's configuration data](#)
- [Methods for manipulating WeblImageFX](#)

- Events for manipulating WebImageFX
- Commands Unique to WebImageFX

To learn how the user interface works, please refer to the **eWebEditPro+XML** User Guide.

Using the WebImageFX Object

Assigning Configuration

Before WebImageFX is displayed, a configuration must be assigned to determine its functionality. This is normally done in the **eWebEditPro+XML**'s configuration XML data.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config product="eWebEditPro">
    .
    .
    <features>
        .
        .
        <mediafiles>
            .
            .
            <imageedit>
                <control src="[WebImageFXPath]/ImageEditConfig.xml" />
            </imageedit>

```

Also, a client script can assign a configuration file to WebImageFX. This is done using the SetConfig method in the Object.

```
objImageEdit.SetConfig(sImageConfigURLorStream);
```

See Also: “[Method: SetConfig](#)” on page 119

Retrieving the Object

To access the feature, a client script must first retrieve the object using the ImageEditor method.

```
var objInstance = eWebEditPro.instances[sEditorName];
var objImageEdit = objInstance.editor.ImageEditor();
```

Checking Availability

When a client retrieves the object, use the IsPresent method to determine if WebImageFX is available.

```
if(false == objImageEdit.IsPresent())
{
    alert("The Image Editor is not available.");
}
```

See Also: “[Method: IsPresent](#)” on page 99

If WebImageFX is available, have the client scripting use the IsVisible method to determine if it is currently displayed to the user.

```
if(false == objImageEdit.IsVisible())
```

```

{
    eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdmfueditimage", "", 0);
}

```

See Also: ["Method: IsVisible" on page 101](#)

Displaying WebImageFX

Because the display of WebImageFX within **eWebEditPro+XML** is a function of **eWebEditPro+XML**, you must use **eWebEditPro+XML**'s command mechanism to display the feature to the user. Use the `cmdmfueditimage` command to make the editor visible.

```
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdmfueditimage", "", 0);
```

You can determine if WebImageFX is already displayed by using the `isVisible` method. If it is, sending the command hides WebImageFX. Here is how to check WebImageFX's display status.

```
if(false == objImageEdit.Visible())
{
    eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdmfueditimage", "", 0);
}
```

See Also: ["Method: IsVisible" on page 101](#)

Controlling WebImageFX

Once the WebImageFX object is obtained and the feature is available, you can control functionality through the object. Below is an example method call that displays the Save As image dialog.

```
objImageEdit.AskSaveAs();
```

See Also: ["Method: AskSaveAs" on page 57](#)

Full Example

Below is a full example that performs all of the object retrieval and error checking to produce the Save As dialog in WebImageFX.

```

function SaveEditedImageAs(sEditorName)
{
    var objInstance = eWebEditPro.instances[sEditorName];
    var objImageEdit = objInstance.editor.ImageEditor();
    if(true == objImageEdit.IsPresent())
    {
        if(true == objImageEdit.IsVisible())
        {
            objImageEdit.AskSaveAs();
        }
    }
    else
    {
        alert("The Image Editor is not available.");
    }
}

```

Adding a Toolbar Button to Launch WebImageFX

By default, the command to launch the feature (cmdmfueditimage) is included within the mediafiles element of the configuration data.

```

<mediafiles>
...
    <!-- The command below will only be enabled when the Ektron
    WebImageFX tool is installed. -->
    <cmd name="cmdmfueditimage" key="freehand" ref="cmdImgEdit" />
...
</mediafiles>

```

Users can execute the command by clicking **Image Editor** from the right-click context menu.

A toolbar button to execute the command is not visible by default. To make it visible, add `<button command="cmdmfueditimage" />` to the toolbar section of **eWebEditPro+XML**'s configuration data.

See Also: "Defining the Toolbar" on page 230

New Configuration Variable

Webimagefx adds an element, `imageedit`, to the `mediafiles` element of **eWebEditPro+XML**'s configuration data. The element specifies the location of the feature's configuration data file. Here is the default value.

```

<mediafiles>
<imageedit>
    <control src="[WebImageFXPath]/ImageEditConfig.xml" />
</imageedit>
</mediafiles>

```

When specified in the configuration, the path expands to include the installation location of WebImageFX on the server:

`http://www.mysite.com/webimagefx/ImageEditConfig.xml`

WebImageFX's Configuration Data

WebImageFX's configuration data is captured during installation into a file, *ImageEditConfig.xml*. The configuration data lets developers manage many aspects of the feature, such as:

- file formats in which graphics can be saved
- whether a user can change an image's format or name
- whether a user can create a new image

See Also: "The Configuration Data" on page 312

The installed version of the file is shown below.

```
<imagedit enabled="true">
<interface name="standard" allowCustomize="false">
    <menu name="editbar" newRow="false"
        showButtonsCaptions="false" wrap="false">
        <caption localeRef="mnuEdit" />
        <button command="cmdtext" />
        <button command="cmdblur" />
    </menu>
</interface>
<operations>
    <valformats enabled="true">
        <imgfmt>image/gif</imgfmt>
        <imgfmt>image/jpg</imgfmt>
        <imgfmt>image/png</imgfmt>
    </valformats>
    <valoutformats>
        <imgfmt>image/jpg</imgfmt>
        <imgfmt>image/png</imgfmt>
    </valoutformats>
    <imgcreate allow="true"/>
    <fmtchange allow="true"/>
    <namechange allow="true"/>
    <command name="cmdtext">
        <image key="imagetext" />
        <caption localeRef="btnText" />
        <tooltiptext localeRef="cmdText" />
    </command>
</operations>
</imagedit>
```

Note that WebImageFX's root element is `<imageedit/>`.

Below is an alphabetical list of elements in *imageedit.xml*, and a link to more information for each one.

NOTE

Since many elements are also used in the standard configuration data, they are explained in that chapter.

Element	For information, see
button	"button" on page 333
caption	"Caption" on page 335
command	"Commands Unique to WebImageFX" on page 593; "command" on page 336
fmtchange	"fmtchange" on page 582
image	"image" on page 342
imgcreate	"imgcreate" on page 583
imgedit	"imgedit" on page 583
imgfmt	"imgfmt" on page 584
interface	"interface" on page 343
menu	"menu" on page 349
namechange	"namechange" on page 584
operations	"operations" on page 585
tooltiptext	"toolTipText" on page 358
valformats	"valformats" on page 586
valoutformats	"valoutformats" on page 587

fmtchange

Determines whether or not the user can change the file format of the image being edited.

See Also: ["imgfmt" on page 584](#)

Element Hierarchy

```
<imgedit>
  <operations>
    <fmtchange>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.
allow	Boolean	true	If this value is true, the user can change the format of the image file being edited.

Example

```
<fmtchange allow="true" />
```

imgcreate

Determines whether or not the user can create a new image.

Element Hierarchy

```
<imgedit>
  <operations>
    <imgcreate>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.
allow	Boolean	true	If this value is true, the user can create a new image.

Example

```
<imgcreate allow="true" />
```

imgedit

Contains all configuration information used by WebImageFX. This is the feature's root element.

Element Hierarchy

```
<imgedit>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within this tag is ignored and WebImageFX is inactive.

Example

```
<imgedit enabled="true">
```

imgfmt

Contains the image formats allowed in WebImageFX. WebImageFX only supports the following graphic file formats: .gif, .jpg, and .png. If you add an unsupported format, it is ignored.

See Also: ["Specifying Image Format" on page 590](#)

This element's values are checked when a user creates a new image or tries to convert an existing image's format.

Element Hierarchy

```
<imgedit>
    <operations>
        <valformats>
            <imgfmt>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.

Example

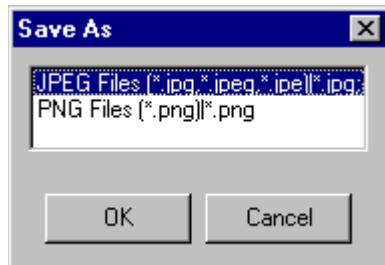
```
<valformats enabled="true">
    <imgfmt>image/gif</imgfmt>
    <imgfmt>image/jpg</imgfmt>
    <imgfmt>image/png</imgfmt>
</valformats>
```

namechange

Determines whether or not the user can change the name of an image file. You would not want to allow this if, for example, changing a file's name might break existing links to it.

Effect of Setting Namechange to False

If namechange is set to false, and the user clicks **Save As** from the File menu, the following dialog appears.



Note that this dialog differs from the normal Save as dialog in the following ways:

- you cannot select a name
- you cannot select a folder

Element Hierarchy

```
<imgedit>
  <operations>
    <namechange>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.
allow	Boolean	true	If this value is false, the user cannot change the name of a file being edited.

Example

```
<namechange allow="true" />
```

operations

Wraps the section that contains the feature settings.

Element Hierarchy

```
<imgedit>
```

<operations>

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.

Example

```

<operations>
    <valformats enabled="true">
        <imgfmt>image/gif</imgfmt>
        <imgfmt>image/jpg</imgfmt>
        <imgfmt>image/png</imgfmt>
    </valformats>
    <imgcreate allow="true"/>
    <fmtchange allow="true"/>
    <namechange allow="true"/>
    <command name="cmdtext">
        <image key="imagetext" />
        <caption localeRef="btnText" />
        <tooltiptext localeRef="cmdText" />
    </command>
</operations>

```

valformats

Contains the list of graphic file formats considered valid by WebImageFX. This tag consists of a series of 'imgfmt' tags.

See Also: "[imgfmt](#)" on page 584

Element Hierarchy

```

<imgedit>
    <operations>
        <valformats>

```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.

Example

```
<valformats enabled="true">
    <imgfmt>image/gif</imgfmt>
    <imgfmt>image/jpg</imgfmt>
    <imgfmt>image/png</imgfmt>
</valformats>
```

valoutformats

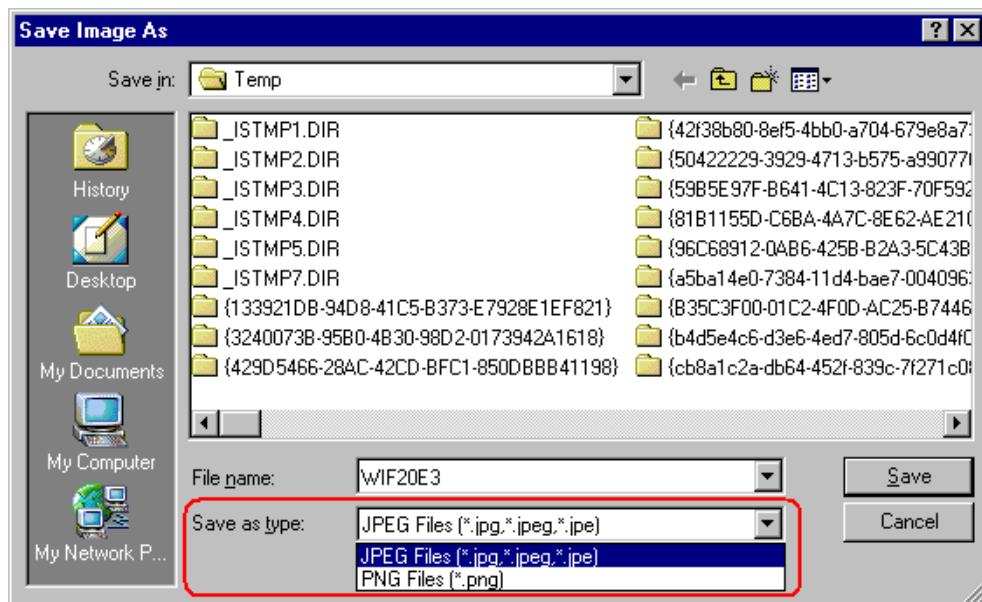
Lets you determine the valid output graphic file formats. If a graphic file format is not listed between these tags, the user cannot save the image in that format.

This tag consists of a series of 'imgfmt' tags. See Also: "["imgfmt" on page 584](#)

How the valoutformats Element is Used

This element may affect the list of choices in the Save As dialog (illustrated below).

See Also: "["Effect of Setting Namechange to False" on page 585](#)



This element is only used if `fmtchange = true`. See Also: "fmtchange" on page 582

Element Hierarchy

```
<imgedit>
  <operations>
    <valoutformats>
```

Attributes

Name	Attribute Type	Default	Description
enabled	Boolean	true	Signals whether this set of data is enabled. If false, all data within the tag is ignored.

Example

```
<valoutformats enabled="true">
  <imgfmt>image/jpg</imgfmt>
  <imgfmt>image/png</imgfmt>
</valoutformats>
```

Image Names

An image file can have two names:

- a remote name, such as ../images/me.png
- a temporary file name, such as c:\temp\me.png

The name passed between the client application and the feature is assigned when the file is first loaded. The assigned name may or may not match the name of the file under which it is saved.

The assigned name is the key that is assigned to the image and references the file being edited. This name may not match the name under which the file is saved.

The assigned name remains constant throughout the editing session for that image, even if the file is saved to a different name. Events and methods will provide this name along with the actual save path and file name.

Here are some examples.

Example 1: Local file is edited then saved locally

loaded file name	c:\images\me.png
temporary file name	c:\images\me.png
saved file name	c:\images\me.png

Example 2: File stored at a URL is saved locally

loaded file name	http://www.yahoo.com/images/me.png
temporary file name	C:\Documents and Settings\username\Local Settings\Temp\me.png
saved file name	c:\windows\temp\me.png

Example 3: File stored in relative path on server is saved locally

loaded file name	../images/me.png
temporary file name	C:\Documents and Settings\username\Local Settings\Temp\me.png
saved file name	c:\windows\temp\me.png

Specifying Image Format

WebImageFX uses the internet standard for specifying a graphic file format. This format allows for easy interchange with HTML and XML. The format is a string composed as follows:

- image
- a slash (/)
- the format designation

Here are some examples:

`image/gif` - the Graphics Interchange format

NOTE Due to licensing issues required of customers and their clients, the GIF format is only supported for a read operation. If a GIF file is modified, it saved in the PNG format.

`image/jpg` - the JFIF compliant format

`image/png` - the Portable Network Graphics format

Separate each file format with a comma. So, a list of formats would look like this:

`image/gif,image/jpg,image/png`

NOTE The `imgformat` element of the configuration data determines which graphic file formats can be used in your system. See Also: "imgfmt" on page 584

Specifying Color Depth

To specify an image's color depth (that is, the number of colors available to the image), specify a *bit depth*. The color depth is derived from the bit depth.

Here are the bit depth values.

Bit depth	Color depth
1	2 colors
4	16 colors
8	256 colors
24	16M colors

Methods to Manipulate WebImageFX

The table below contains all methods available to manipulate WebImageFX. Following the table is a detailed description of each method.

The table's columns indicate which methods are available to the client and the server. As you can see, all methods are available via the user interface, but only some can be executed programmatically on the server.

NOTE

If you want to process images on the server, its operating system must be Windows NT Server, Windows 2000 Server, or Windows XP Server.

Method	Client function	Server function	For more information, see page
AskOpenFile	X		57
AskSaveAs	X		57
AskSelectColor	X		57
ConvertImage	X	X	65
CreateNew	X	X	67
EditFile	X	X	71
EditFromHTML	X		72
EnableCreation	X		76
EnableFormatChange	X		76
EnableNameChange	X		77
ErrorClear	X	X	78
ErrorDescription	X	X	78
ErrorValue	X	X	79
ExecCommand	X		80
GetImageInformation	X	X	89
GetValidFormats	X		92
ImageEditor	X		94

Method	Client function	Server function	For more information, see page
IsDirty	X		96
IsPresent	X		99
IsVisible	X		101
LoadedFileName	X		105
PublishHTML	X		111
Save	X	X	116
SaveAs	X	X	117
SavedFileName	X		117
SetConfig	X		119
SetLocale	X		123
SetValidFormats	X		125
Thumbnail	X	X	127

Events to Manipulate WebImageFX

Events are called by WebImageFX into a client script, which defines how to accept an event from WebImageFX. As a result, when something happens in WebImageFX, it calls the event. The client script receives this call and can react to the notification.

The table below contains all events available to WebImageFX. Following the table is a detailed description of each event.

All events are available on the client only - none is available on the server.

Event	For more information, see page
EditCommandComplete	183
EditCommandStart	183

Event	For more information, see page
EditComplete	184
ImageError	184
LoadingImage	185
SavingImage	185

Commands Unique to WebImageFX

The following commands are available within WebImageFX's configuration data.

See Also: "Commands" on page 195

Command Name	Function
cmdblur	Blurs or softens an image
cmdbrightness	Changes an image's brightness
cmdchoosecolor	Assigns color of annotation before user inserts it
cmdchoosefont	Assigns color of text before user inserts it
cmdcolordepth	Changes the number of colors available to an image
cmdcontrast	Changes the difference between light and dark areas of an image
cmdcopy	Copies selected text into the copy buffer
cmdcreatenew	Creates a new image
cmdcrop	Removes everything outside the selected area of an image
cmddelete	Deletes selected area of an image
cmddimensions	Lets the user modify an image's width and height
cmddelete	Deletes selected text

Command Name	Function
cmdexit	The edited file is saved, WebImageFX closes, and eWebEditPro+XML reappears with the edited image.
cmdfreehand	Draws a line in any shape that the user wants. See Also: "The IData Parameter" on page 595
cmdfullview	Displays image at full size
cmdhorizflip	Reverses an image horizontally left to right
cmdimageinfo	Displays information about an image
cmdline	Draws a straight line. See Also: "The IData Parameter" on page 595
cmdopen	Displays standard Open File dialog, which lets user select an image to edit
cmdoval	Draws an oval. See Also: "The IData Parameter" on page 595
cmdpastenew	Pastes the contents of the copy buffer into a new image file
cmdpointer	Lets user click on an annotation to select it
cmdpolygon	Draws a polygon. See Also: "The IData Parameter" on page 595
cmdrectangle	Draws a rectangle. See Also: "The IData Parameter" on page 595
cmdredo	Executes the action that occurred right before the user executed cmdundo
cmdrotate	Turns an image a specified number of degrees
cmdsave	The first time an image is saved, this command displays the standard Save File dialog box, which prompts the user to save the image to a selected file location. Subsequently, this command saves the current version of the image to that file location.
cmdsaveas	Displays the standard Save File dialog box, which prompts the user to save the image to a selected file location
cmdselect	Selects an area of an image. The user can then perform actions on the area, such as blur and delete. See Also: "The IData Parameter" on page 595
cmdsharpen	Sharpens edges within an image

Command Name	Function
cmdtext	Inserts text onto the image. See <i>Also:</i> "The IData Parameter" on page 595
cmdtwainacquire	Performs a single page scan. Before scanning, the user must select a source using the Twain Source command.
cmdtwainsource	Selects a source for acquiring an image, such as a scanner or digital camera
cmdundo	Reverses the most recent command
cmdvertflip	Flips an image vertically top to bottom
cmdzoomin	Increases an image's magnification
cmdzoomout	Decreases an image's magnification

The IData Parameter

Several commands are toggle commands. This means that when they are turned on, they stay on until turned off. As examples, bold and italic are toggle commands.

Toggle commands use the IData parameter to determine their state. If IData = 0, the command is turned off. If IData is non-zero, it is turned on.

The following commands use the IData parameter.

- cmdfreehand
- cmdline
- cmdoval
- cmdpolygon
- cmdrectangle
- cmdselect
- cmdtext

Client Script Interface for Automatic File Upload

This section describes the API that lets client scripts control the automatic upload of image files in WebImageFX.

Initializing the Automatic Upload

The Automatic Upload is configured in the WebImageFX configuration. Specifically, the `transport` and `autoupload` elements in the configuration determine how the feature functions when the editor first loads into a Web page.

As the page processes its information, you may want it to modify or activate items in the upload functionality. The Automatic File Upload interface provides the methods and properties to let you do this.

Interface Retrieval

To retrieve the interface that controls the upload functionality, use the core JavaScript's `instances` object array.

```
var objEditor = WebImageFX.instances[g_sEditorName];
var objAutoUpload = objEditor.editor.AutomaticUpload();
```

Then, access the functionality through this object interface.

```
objAutoUpload.setProperty("TransferMethod", sTransferMethod);
objAutoUpload.AddFileForUpload(sMyFileName, sMyDescription);
objAutoUpload.AddNamedData(sMyFileName, "username", sMyUserName);
```

Use the following command mechanism to initiate the upload. It confirms the upload with the user and sends each file to the server.

```
WebImageFX.instances[g_sEditorName].editor.ExecCommand("cmdmfuuploadall", "", 0);
```

Properties

The configuration data initially sets all property values. You only need to modify them to change how the startup configuration operates.

AllowUpload

See ["Property: AllowUpload" on page 148](#)

WebRoot

See ["Property: WebRoot" on page 147](#)

ValidExtensions

See "Property: ValidExtensions" on page 147

TransferRoot

See "Property: TransferRoot" on page 147

Port

See "Property: Port" on page 149

LoginRequired

See "Property: LoginRequired" on page 147

LoginName

See "Property: LoginName" on page 154

Password

See "Property: Password" on page 147

TransferMethod

See "Property: TransferMethod" on page 157

ServerName

See "Property: ServerName" on page 146

Methods**GetFileDescription(FileName)****Description**

Returns the description of the specified file. If the file does not exist, the return value is an empty string.

Return Type

String

Parameters

FileName - The file name which has the returned description applied to it.

SetFileDescription(FileName, Description)

See "Method: SetFileDescription" on page 121

ReadResponseHeader()

See "Method: ReadResponseHeader" on page 112

AddNamedData(FileName, DataName, DataValue)

See "Method: AddNamedData" on page 54

ReadNamedData(FileName, DataName)

See "Method: ReadNamedData" on page 112

RemoveNamedData(FileName, DataName)

See "Method: RemoveNamedData" on page 115

GetFileStatus(FileName)

See "Method: GetFileStatus" on page 87

SetFileStatus(FileName, Status)

See "Method: SetFileStatus" on page 122

ReadUploadResponse()

See "Method: ReadUploadResponse" on page 113

UploadConfirmMsg(Message, Title)

See "Method: UploadConfirmMsg" on page 131

SetFieldValue(DataName, DataValue)

See "Method: SetFieldValue" on page 121

GetFieldValue(DataName)

See "Method: GetFieldValue" on page 85

RemoveFieldValue(DataName)

See "Method: RemoveFieldValue" on page 114

AddFileForUpload(LocalFileName, Description)

See "Method: AddFileForUpload" on page 51

ListFilesWithStatus(Status, Delim)**Description**

Returns a delimited list of all files with the specified status.

Return Type

String

Parameters

Status - The status to use to retrieve the files. A status can be a combination of many status values, so the status is returned as bits set in a long value.

Delim - The delimiter to use between file entries in the returned string.

RemoveFileForUpload(LocalFileName)

See "["Method: RemoveFileForUpload"](#) on page 114

Property Setting Methods

Under some versions of Netscape, the properties cannot be accessed directly. To circumvent this problem, you can use the following methods to ensure that the properties can be set from all browsers. These standard methods are included with all objects in the Ektron family of editors.

- ["Method: setProperty"](#) on page 124
- ["Method: getProperty"](#) on page 90
- ["Method: getPropertyString"](#) on page 91
- `getPropertyInteger(Name As String) As Long`
- ["Method: getPropertyBoolean"](#) on page 90

eWebEditPro+XML's XML Features

eWebEditPro+XML provides two methods for implementing XML functionality. They are contrasted on the table below.

Method	Description	For more information, see
Original	<p>The developer designs the schema and transformations. The end-user uses those schema and transformations.</p> <p>Key elements are</p> <ul style="list-style-type: none">Custom XML tags can be mixed inline with HTML content in a documentConfiguration settings determine the appearance and behavior of custom XML tags within the user interface. An XML schema can further configure the interface.Provides facilities to automatically run XSLTs during input and output from eWebEditPro+XMLThe user can rearrange, delete, duplicate, and modify inline XML tags and place HTML content within an XML tag	"Original XML Functionality" on page 644
Data Designer	<p>The user generates the schema and transformations. This allows them to determine which elements to include and how to transform them.</p> <p>Key elements are</p> <ul style="list-style-type: none">two additional user interface modes:<ul style="list-style-type: none"><i>Data Design</i> mode, which is used to define the structure, validation rules, and user interface for an XML document type<i>Data Entry</i> mode for creating an XML document conforming to the “data design” document typeThe user is presented with a “Smart Form” interface that enforces data design rules	"Supporting the Data Designer" on page 601

Supporting the Data Designer

The Data Designer feature is primarily intended for end users. Therefore, most of the information about it is in the “Using the Data Designer” chapter of the **eWebEditPro+XML** User Guide. However, this section explains changes to the configuration data and other background information that are needed to operate the Data Designer via the following topics.

- “Changes to eWebEditPro+XML’s Configuration” on page 601
- “Data Designer Commands” on page 603
- “Retrieving and Loading Data Designer Content” on page 607
- “Selecting an External File Link” on page 612
- “Indexing the Fields of a Data Design Document” on page 613
- “Customizing Validation Options” on page 615
- “Methods, Properties, and Events that Provide Access to Data Designer Fields” on page 632
- “Letting Users Execute a Custom Function on a Data Design Screen” on page 634
- “Specifying Items in a Select List Field” on page 636

NOTE

When using the Data Designer feature, the following browsers are supported: Netscape 6.2 or later; Internet Explorer 5.5 or later.

Changes to eWebEditPro+XML’s Configuration

Because this feature requires changes to **eWebEditPro+XML**’s configuration, Ektron supplies two additional configuration files, which are installed to the same directory to which **eWebEditPro+XML** is installed.

- *configdatadesign.xml* - affects pages on which you design a data entry screen
- *configdataentry.xml* - affects pages on which users enter content into the designed screen

Any page that uses the Data Designer feature must refer to one of these files. Below is an example of how to invoke configdatadesign.xml on a page that hosts the editor.

```
eWebEditPro.parameters.config = ".../.../.../configdatadesign.xml";
```

NOTE

If **eWebEditPro+XML** starts in design mode with configdatadesign.xml, and the user clicks the button to preview data entry, the configdatadesign.xml is still active because it is the configuration XML data assigned to the editor.

How These Configuration Files Differ from config.xml

The two configuration files differ from config.xml in the following ways.

- They include toolbars whose buttons are used to insert Data Design fields on a screen. The toolbars are found between the <datadesign> and <dataentry> tags.
- They remove standard toolbar buttons that are invalid when using the Data Designer, such as Edit in Word.
- configdatadesign.xml also includes these unique elements and attributes.
 - the <datadesign> element includes information about the following

Topic	For information, see
toolbar buttons	"Data Designer Commands" on page 603
indexing	"Indexing the Fields of a Data Design Document" on page 613
sample calculation formulas	"The <cmddsgcalc> Command" on page 605
provides control over what the presentation XSLT produces	

- view attribute of the viewsas element, view. See Also: ["<Viewsas> Attribute: View" on page 602](#)
- publishinvalid attribute of the <standard> element. See Also: ["Saving Invalid Documents" on page 625](#)
- the <validation> element and its subelements. See Also: ["Customizing Validation Options" on page 615](#)

<Viewsas> Attribute: View

The viewsas element's attribute, view, has the following possible values. For example:

```
<viewsas publish="xhtml" mode="body" unicode="false" view="datadesign">
```

Value	eWebEditPro+XML appears in
datadesign	Data Design mode. In this mode, the data design toolbars appear, and normally visible XML tags are suppressed. This mode allows business users to create data entry screens.
dataentry	Data Entry mode. In this mode, the data design toolbars do not appear, and XML tags that are normally visible are suppressed. This mode allows business users to complete data entry screens.
wysiwyg	WYSIWYG mode. The editor appears as it does without the Data Designer.
source	View Source mode. The user sees the document's XML structure instead of the WYSIWYG view.

Data Designer Commands

See Also: "cmd" on page 339

Command	Icon	Description	See also this section in User Guide chapter "Using the Data Designer"
cmddatadesign	 design	Switches to Data Design mode. Only active when editor is in Data Entry mode.	
cmddataentry	 fillin	Switches to Data Entry mode. Only active when editor is in Data Design mode.	
cmddsgfieldset	 fieldset	Groups related fields, which can be surrounded by a box and have a caption.	Group Box

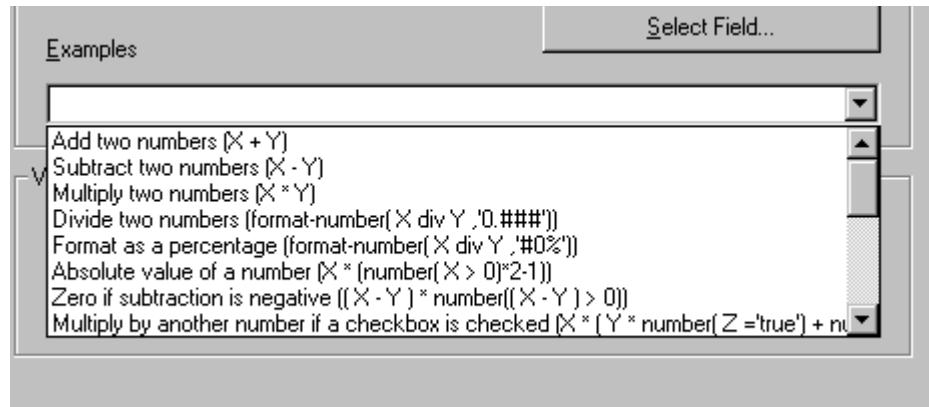
Command	Icon	Description	See also this section in User Guide chapter "Using the Data Designer"
cmddsgboolean	 checkbox	User response is either checked or unchecked	Checkbox
cmddsgplaintext	 textfld	Free text field; user cannot format text	Plain Text
cmddsgricharea	 richtextfld	Free text field; user can format text using the editor	Rich Area
cmddsgchoices	 choices	Values appear on screen, and user selects appropriate ones.	Choices
cmddsglistcontrol	 dropdown	Same as Choices except values appear in drop-down box.	Select List
cmddsgprop	 properties	Lets you change the field properties. Select a field then click this button.	
cmddsgcalc	 calculation	Insert/edit Calculated field. See Also: "The <cmddsgcalc> Command" on page 605	Calculated Field
cmddsgcalendar	 calendar	Insert/edit Calendar field.	Calendar Field
cmddsgimageonly	 image only	Insert/edit Image only field.	Image Only Field
cmddsgfilelink	 file link	Insert/edit file link field.	File Link Field
cmdvalidxsd	 validate	Validate. (Available in Design mode.) See Also: "The Validate Commands" on page 606	
cmdvalidate	 validate	Validate. (Available in Data Entry mode.) See Also: "The Validate Commands" on page 606	
cmdvalidayt	 validayt	Validate As You Type. Toggle button. (Available in Data Entry mode.)	

The <cmddsgcalc> Command

The <cmddsgcalc> command has an optional child element, <selections name="examples">, that you can use to define a list of XPath examples. For example:

```
<command name="cmddsgcalc">
  <image key="calculation"/>
  <caption localeRef="cmdCalcFld"/>
  <tooltiptext localeRef="cmdCalcFld"/>
  <selections name="examples">
    <listchoice value="sum({X})" ...>
    <listchoice value="sum({X}|{Y}|{Z})" ...>
    <listchoice value="{X} * {Y}" ...>
    <listchoice value="{X} div {Y}" ...>
  </selections>
</command>
```

These example appear in the Examples dropdown list of the Calculated Field dialog.



Controlling the Presentation XSLT

The <xsltoutput> element, located under the <datadesign> element of the Data Designer configuration data, lets you control the output generated by the presentation XSLT. This element is a convenience, not a necessity, since the presentation xslt is not required.

Here is an excerpt from configdatadesign.xml.

```
<datadesign ...>
:
<!-- mode="whole|body" method="xml|html|text" encoding="string" omit-xml-
declaration="yes|no" indent="yes|no" standalone="yes|no" cdata-section-elements="qnames"
media-type="string" -->
<xsltoutput name="datapresentationxslt" mode="body" method="xml" encoding="utf-8"
omit-xml-declaration="yes"/>
</datadesign>
```

The following table describes the attributes listed in the above excerpt.

NOTE Except for mode, the following are standard XSLT attributes. See <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/xmrefxsloutputelement.asp> for a complete description.

Attribute	Possible values/ type	Description
mode	<ul style="list-style-type: none">● whole● body <p>default=body</p>	Controls whether the XSLT outputs an entire HTML document or a fragment suitable for inclusion in a larger HTML document
method	<ul style="list-style-type: none">● xml● html● text <p>default=xml</p>	Determines whether content conforms to standard XML/XHTML, HTML, or no standard. This is the XSLT standard <xsl:output method> attribute.
encoding	string default=utf-8	The XSLT standard <xsl:output encoding> attribute.
omit-xml-declaration	boolean default=yes	The XSLT standard <xsl:output omit-xml-declaration> attribute.
indent	boolean default=yes	The XSLT standard <xsl:output indent> attribute.
standalone	boolean default=yes	The XSLT standard <xsl:output standalone> attribute.
cdata-section-elements	default=qnames	The XSLT standard <xsl:output cdata-section-elements> attribute.
media-type	string	The XSLT standard <xsl:output media-type> attribute.

The Validate Commands

The Validate commands (`cmdvalidxsd` and `cmdvalidate`) validate the content. Depending on the mode, the definition is slightly different.

See Also: "Customizing Validation Options" on page 615

Design Mode

In Design mode (`cmdvalidxsd`), the design is validated to ensure the schema is valid. A schema is invalid if the design has more than one field with the same name in the same group.

Data Entry Mode

In Data Entry mode (`cmdvalidate`), the content is validated in two steps.

1. Validate the data in each field.
2. Validate the whole XML document against the schema.

In Data Entry mode, when you move the cursor out of a field, that field's data is validated whether or not “Validate As You Type” (VAYT) is set. If the VAYT button is depressed, the data in all fields is occasionally validated. This takes some time, but should not be obtrusive. The whole XML document is *not* validated against the schema because it takes too long.

While an individual field's data is checked when the user clicks away, VAYT is still needed to check relationships between fields. For example, two Plain Text fields both collect a number. You can determine that the number in the first field must be less than the number in the second field. Because changing the data value in the first field may make the second field invalid, VAYT checks the validity of all other fields.

Retrieving and Loading Data Designer Content

eWebEditPro+XML's API lets you load more than one content field (that is, hidden form field) into the editor, and retrieve more than one piece of content from the editor.

In Data Entry mode, the editor typically receives an XML document (your content), an XSLT to control the display, and a schema to validate the data.

In Data Design mode, when saving content out of the editor, the editor typically returns a default XML document (default content), a data entry XSLT (to determine the appearance of the data entry screen), a schema to validate the data, a presentation XSLT (to determines a default format for the content), and a design page (essential information to be saved to the database).

Typical Use of Content Types

Design Mode

1. Set `designpage` (or empty string if creating a new design)
2. Get `designpage`
3. Get `datadocumentxml`
4. Get `dataentryxslt`
5. Get `datapresentationxslt`
6. Get `dataschema`

Data Entry Mode

1. Set datadocumentxml
2. Set dataschema
3. Set dataentryxslt
4. Get datadocumentxml

To display a document in a Web browser (without involving the editor), transform “datadocumentxml” using “datapresentationxslt”.

Content Types

The types of content that you can work with are listed in ["Content Type Categories" on page 570](#). The following content types are especially helpful when getting and setting content.

- datadesignpackage
- datadocumentxml
- dataentryxslt
- datapresentationxslt
- dataschema
- designpage
- dataindex

Integration with Web Application Servers and Languages

The API that lets you load more than one content field into the editor is implemented for [ASP](#), [ColdFusion](#), [JSP](#), [PHP](#), and client-side [JavaScript](#). It can be easily implemented in other server script languages.

ASP

```
eWebEditProField(EditorName, FieldName, SetType, GetType, ContentHtml)
```

For information about the parameters, see ["Description of the Function Parameters" on page 611](#)

Example

Design mode

```
<% =eWebEditProEditor("editorXml", "100%", "85%", "") %>
<% =eWebEditProField("editorXml", "designpage", "designpage", strDesignPage) %>
<% =eWebEditProField("editorXml", "dataschema", "", "dataschema", "") %>
<% =eWebEditProField("editorXml", "datadocumentxml", "", "datadocumentxml", "") %>
<% =eWebEditProField("editorXml", "dataentryxslt", "", "dataentryxslt", "") %>
<% =eWebEditProField("editorXml", "datapresentationxslt", "", "datapresentationxslt", "") %>
```

Data entry mode

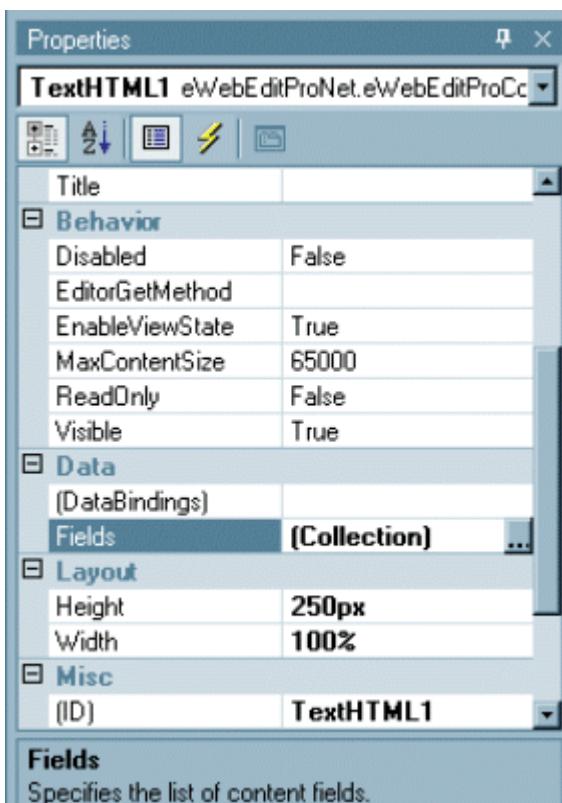
```
<% =eWebEditProEditor("edit_xml", "100%", "85%", "") %>
<% =eWebEditProField("edit_xml", "dataentryxslt", "dataentryxslt", "", strDataEntryXSLT) %>
<% =eWebEditProField("edit_xml", "dataschema", "dataschema", "", strDataSchema) %>
<% =eWebEditProField("edit_xml", "datadocumentxml", "datadocumentxml", "datadocumentxml",
strDataDocumentXML) %>
```

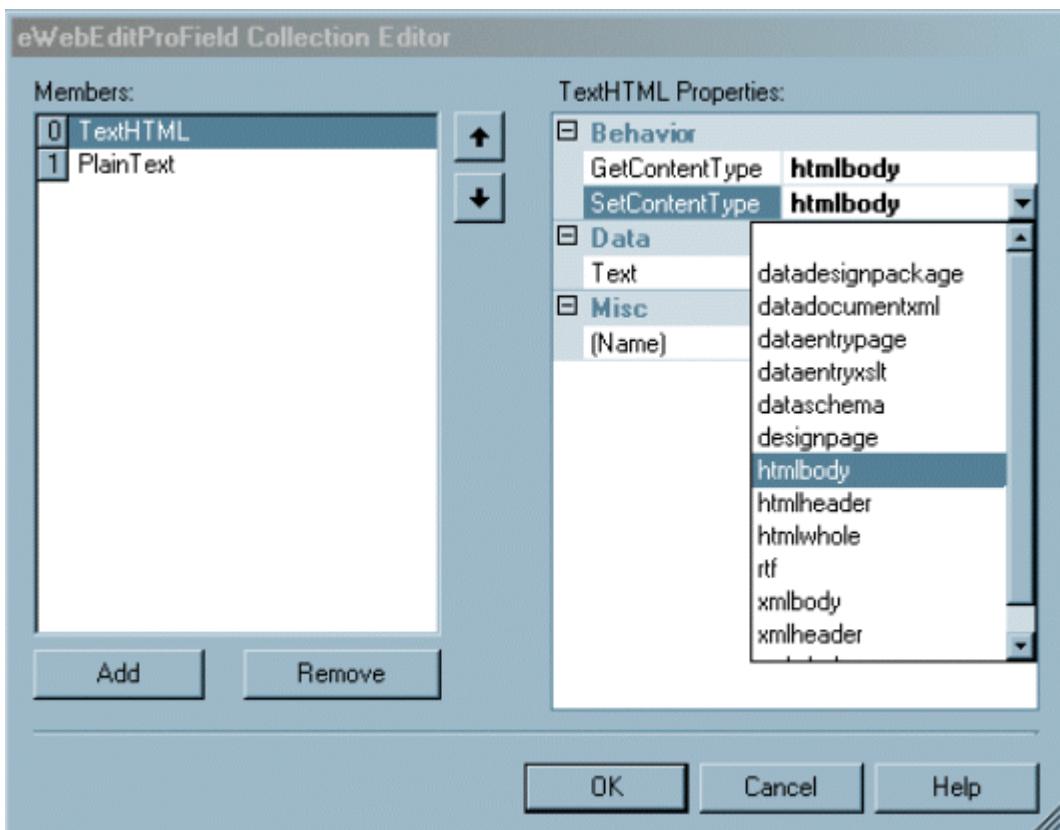
ASP.NET

Use Visual Studio .Net properties to define the content fields. Content should be read from the field's text property. This is typically done in the code behind.

To access the text field, click the button next to the **Collection** response to the **Data > Fields** property (illustrated below).

To get of list of which fields to define, see "[Typical Use of Content Types](#)" on page 607.





ColdFusion

```
<cfmodule template="ewebeditprofield4.cfm" EditorName="" FieldName="" SetType="" GetType="" Value="">
```

For information about the parameters, see ["Description of the Function Parameters" on page 611](#).

Example

Design Mode

```
<cfmodule template="../../../../ewebeditpro5.cfm" EditorName="editorXml" Width="100%" Height="75%" Config="../../../../configdatadesign.xml">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="editorXml" FieldName="designpage" SetType="designpage" GetType="designpage" Value="#designpage#">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="editorXml" FieldName="dataentryxslt" GetType="dataentryxslt">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="editorXml" FieldName="datapresentationxslt" GetType="datapresentationxslt">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="editorXml" FieldName="dataschema" GetType="dataschema">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="editorXml" FieldName="datadocumentxml" GetType="datadocumentxml">
```

Data Entry Mode

```
<cfmodule template="../../../../ewebeditpro5.cfm" EditorName="edit_xml" Width="100%" Height="75%" Config="../../../../configdataentry.xml">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="edit_xml" FieldName="dataentryxslt" SetType="dataentryxslt" Value="#dataentryxslt#">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="edit_xml" FieldName="dataschema" SetType="dataschema" Value="#dataschema#">
<cfmodule template="../../../../ewebeditprofield4.cfm" EditorName="edit_xml" FieldName="datadocumentxml" SetType="datadocumentxml" GetType="datadocumentxml" Value="#datadocumentxml#">
```

JSP

```
String eWebEditProField(String EditorName, String FieldName, String SetType, String GetType, String ContentHtml)
```

For information about the parameters, see ["Description of the Function Parameters" on page 611](#).

PHP

```
eWebEditProField( $EditorName, $FieldName, $SetType, $GetType, $ContentHtml, $CharacterSet="ISO-8859-1" )
```

For information about the parameters, see ["Description of the Function Parameters" on page 611](#).

NOTE CharacterSet is the same as CharacterSet in the PHP eWebEditProEditor function.

JavaScript (eWebEditPro Object)

```
defineField(editorName, name, setType, getType, parameters)
```

For information about the parameters, see ["Description of the Function Parameters" on page 611](#).

NOTE You must define an HTML form field (typically `input type=hidden`) on the HTML page prior to the script that calls this method. See the example below.

Description of the Function Parameters

Parameter	Description
EditorName	The instance name of the editor.
FieldName	The name of the hidden content field.

Parameter	Description
SetType	Identifies the type of the content to load into the editor. Set to an empty string ("") if the content is only returned from the editor, but not loaded into the editor.
GetType	Identifies the type of content in the editor to save. Set to an empty string ("") if the content is only loaded into the editor, but not posted back.
Content	When loading the page, this is the initial content to load into the editor. When posted, this is the content returned from the editor.
parameters (optional, typically not used)	Same as those documented in the eWebEditPro.create() method. <i>See Also:</i> "Method: create" on page 66

Remarks

- The EditorName may be the same as the FieldName. The names are case-sensitive.
- The SetType and GetType values may be empty strings, have the same value, or different values.
- Fields that have a SetType but an empty GetType are cleared prior to posting back to the server. That is, they are empty strings. This reduces the amount of data posted back.
- Fields can be defined before or after the editor is defined.

See Also: "Content Upload" on page 563

Selecting an External File Link

You can set up a link-only data design field to use an *external link*, that is, a link from outside **eWebEditPro+XML**. Use this feature when specifying an external script for file selections, such as with a Content Management System.

To set up a link-only field to use an external link, set these two values for a file link in the media information object.

```
m_objMediaFile.SrcFileLocationName = strFilePathAndName;
m_objMediaFile.FileTitle = strFileDescription;
```

Then, send the media command.

```
objEditor.editor.ExecCommand( "cmdmfumedia", "", 1);
```

The third parameter value (1) displays the link dialog for the selection. If the parameter were 0, the image properties dialog would appear.

Indexing the Fields of a Data Design Document

After you integrate **eWebEditPro+XML** into a Content Management System (CMS), such as Ektron's CMS300, you probably want the ability to search on specific fields. For example, a CMS search page could let the user select any combination of these criteria

- people whose last name begins with the letter A through D
- accounts whose balance exceeds \$1000
- all transactions within the last 30 days

eWebEditPro+XML lets you index selected fields, which can then be searched by a CMS. Here is an overview.

1. When creating the field (or later editing it) in the Data Designer, you determine if it should be indexed. Only indexed fields can be searched.
2. Use the GetContent method to retrieve index information (field name and data type) for the indexed fields in a document.
3. Use this information in your CMS system to create a search page.

Setting an Index Flag on Selected Fields

Before you can determine which fields to index, enable the indexing capability in the configuration data. To do so, add `<indexing enabled="true">` to the `<datadesign>` element, so that it looks like this. To make the **Indexed** field appear on the dialogs, set `<visible>` to true.

```
<datadesign>
  <indexing enabled="true" visible="true">
</datadesign>
```

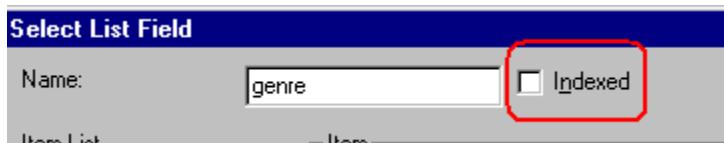
WARNING!

Remember that the Data Designer uses a unique configuration file. By default, it is configdatadesign.xml.

After you enable indexing, an **Indexed** field appears on the dialog box of all field types except Group Box. (Group box fields do not contain data.)

NOTE

If you set the `enabled` attribute to `false`, the **Indexed** field appears but is grayed out.



To index a field, check the **Indexed** field. Only Indexed fields appear in the index information file.

The Index Information File

To retrieve an XML document's index information, use the GetContent Method and specify the dataindex content type. (See Also: "Method: GetContent" on page 85)

The CMS can retrieve the index information file when the XML data design is saved. The content returned via GetContent("dataindex") is an XML formatted document with the following information for each indexed field.

- field's data type (See Also: "Data Types" on page 614)
- xpath index to the field

For example

```
<indexable>
<xpath type="string">/root/ZipCode</xpath>
</indexable>
```

Data Types

The data type for some fields in the Index Information File is determined by the field's validation. Below is a table of field types, validation, and the resulting data type.

NOTE The ImageOnly and FileLink fields return two xpahs.

Field Type	Validation	Data Type
Checkbox		boolean
Richarea		anyType
Choices		selection
Select List		selection
ImageOnly		string (the <Alt> attribute text)
ImageOnly		URL (the link to the image)
FileLink		string (the text to which the link is applied)
FileLink		any URL (the link to the file)
Calculated	No validation	string

Field Type	Validation	Data Type
Calculated	Non-negative number	decimal
Plain Text	No validation	string
Plain Text	Cannot be blank	string
Plain Text	Non-negative whole number or blank	string
Plain Text	Non-negative whole number (required)	nonNegativeInteger
Plain Text	Percent 0-100 (required)	nonNegativeInteger
Plain Text	Social Security Number (U.S.)	string
Plain Text	Email address	string
Plain Text	Zip code (U.S.)	string
Plain Text	Custom	string, anyURI, integer, decimal, or double

Customizing Validation Options

In the configdatadesign.xml file, you can specify a set of validation options for plain text and calculation field types. (Data in other field types cannot be validated.) For these field types, you can assign standard and custom validation checks. The checks are applied when data is inserted into one of the field types, and when the user tries to save a Data Design document whose fields have validation attributes.

You can modify the standard options and enter your own criteria for each field. You can also establish dependencies between fields. For example, a value is only required for a field if a certain Checkbox field is checked.

Validation Elements in the Configuration Data

The default version of the Data Designer configuration file includes standard validation options for plain text and calculation field types.

```
<validation name="calculation">
    <choice name="none" treeImg="text">
        <caption localeRef="dlgNV8n" />
        <schema datatype="string" />
```

and

```
<validation name="plaintext" visible="false">
  <choice name="none" treeImg="text">
    <caption localeRef="dlgNV8n" />
```

The validation sections let you control the drop-down list of validation choices for a field type (for example, Plain Text). Here is an overview of that section of the configdatadesign.xml file.

```
<datadesign>
  <validation> (0 or more)
    <choice> (0 or more)
      <caption>
      <schema>
      <calculate>
    <regexp> OR
      <script> OR
      <>xpath>
      <validate>
        <regexp> OR
        <script> OR
        <>xpath>
    <errormessage> (optional)
```

Details are provided below.

Validation Attributes

Attribute	Description	Possible values
visible	Controls whether the Validation field appears on the field's Properties dialog.	true, false
enabled	Controls whether the Validation field is active or "grayed out" on the field's Properties dialog. If set to true , the Validation field is active; if false , it appears but is grayed out.	true, false
name	The field type to which the validation configuration data applies. A separate <validation> element must exist for each field type.	plaintext, calculation

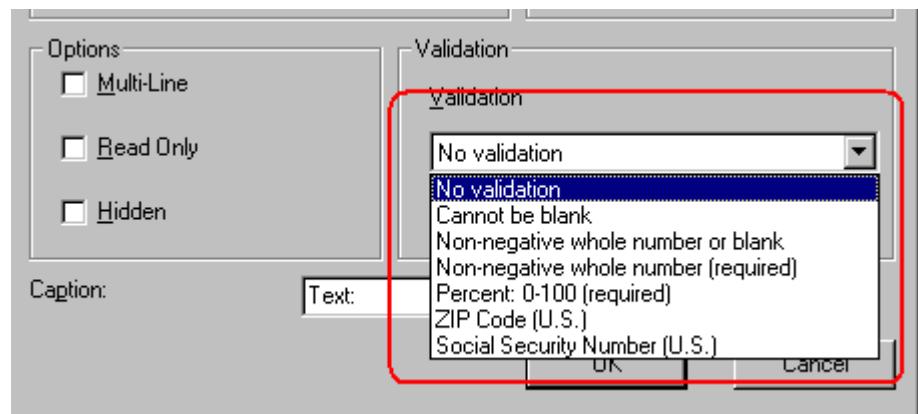
Validation Sub-elements

Element	Description	For more information, see
Choice	The calculation or validation expression for standard fields	"Choice Sub-element" on page 617
Custom	The calculation or validation expression for custom fields	"Defining Custom Validation" on page 621

Choice Sub-element

Every item in the validation drop-down list must be defined within a set of <choice> tags. Within the <choice> tags, you define a caption, a schema, and either a calculation or validation expression.

The default validation choices appear below.



Choice Attributes

Attribute	Description
name	Assign a new name to each choice.
treelmg	The icon to display for this field in the Select Field or Group dialog. See "Icons on the Select Field or Group Screen" on page 626. Unlike toolbar icons, you cannot create your own icons.

Choice Sub-Elements

Element	Description	For more information, see
caption	Defines the displayed text for this choice. The attributes and description are the same as <caption> elements for commands.	"Caption" on page 335
schema	Defines a W3C XML schema (WXS) definition for this choice. The definition may be a simple type defined by the datatype attribute and/or a WXS fragment.	"Defining a Schema Fragment" on page 618
calculate	Defines an expression which normalizes a value prior to checking validation	"Defining a Calculation" on page 619
validate	Defines an expression that determines if a value is valid	"Defining Validation" on page 620

Defining a Schema Fragment

When defining a schema fragment, the "xs" namespace prefix is required for WXS tags. Also, the WXS fragment must be valid for inclusion in an xs:restriction (simple type). That is,

```
<xs:simpleType>
  <xs:restriction>
    ...fragment...
  </xs:restriction>
</xs:simpleType>
```

Simple Data Types

The simple data types are defined by W3C XML Schema definition language 1.0.

NOTE

The datatype value should *not* include a namespace prefix. For example datatype="string" is correct; datatype="xs:string" is incorrect

Examples

- Using only a datatype attribute

```
<schema datatype="string"/>
```
- Using only a schema fragment

```
<schema>
  <xs:simpleType>
    <xs:union memberTypes="xs:nonNegativeInteger">
      <xs:simpleType>
```

```

<xs:restriction base="xs:string">
    <xs:length value="0"/>
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
</schema>

```

- Using a datatype attribute *and* a schema fragment

```

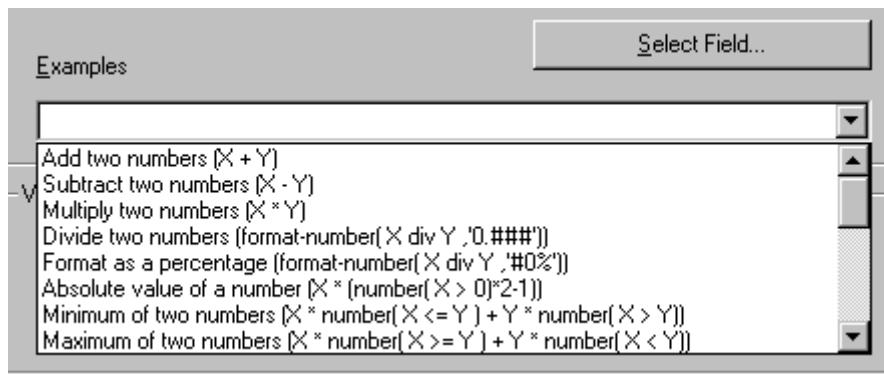
<schema datatype="string">
    <xs:minLength value="1"/>
</schema>

```

Defining a Calculation

Use the `<calculate>` element to define an expression that normalizes a value prior to checking validation. For example, a calculation can truncate digits in a decimal, remove excess white space, or capitalize text. The expression must return a value of the same data type and format as the original value.

Several sample calculations are delivered with **eWebEditPro+XML**. They appear when the user clicks the **Examples** drop-down on the Calculated Field dialog. These calculations are explained in the “Explanation of Examples” section of the “Using the Data Designer” chapter of the User Guide.



You can only define one expression for `<calculate>` and `<validate>`.

The expression element choices are listed below. Use whichever language is easiest.

- Regular Expression (JScript)
- JavaScript
- XPATH

Expression	Description
Regular Expressions (JScript)	

Expression	Description
<regexp>	<p>Defines a regular expression supported by JScript. You can define a regular expression <i>either</i> between the tags or using attributes.</p> <p>If between the tags, the expression must begin with a slash (/) character. The g, i, and m flags are allowed. For example: <regexp>/\S+</regexp>.</p>
<regexp pattern="exp-pattern">	A regular expression pattern (without the "/" chars).
<regexp global="true false">	Specifies whether the pattern matches only the first occurrence or all occurrences within the text. This corresponds to the g flag.
<regexp ignorecase="true false">	Specifies if the match is case-sensitive. This corresponds to the "i" flag.
<regexp multiline="true false">	Specifies if the match, when using ^ and \$, is applied to each line in text that has multiple lines. This corresponds to the "m" flag.
<regexp wholeline="true false">	Specifies whether the pattern applies to the whole text or not. This is the same as placing "^(" at the beginning of the pattern, and ")"\$" at the end of the pattern.
JavaScript	
<script value="javascript-expression">	<p>Specifies a JavaScript expression to be evaluated. The field's value is available in a property named 'this.text'. For example</p> <pre><script value="this.text.toUpperCase()"/></pre>
XPATH (See Also: " Learn More about XPath " on page 628)	
<xpath select="xpath-expression">	Specifies an XPath expression to be evaluated. The field's value is available using ".".

Defining Validation

Use the <validate> element to define an expression that determines if a value is valid. The expression must return a Boolean (true or false) result. For example:

```
<validate>
    <regexp>/^\d+$/</regexp>
</validate>
```

To construct the <validate> element, use the same expression element options as <calculate>. See Also: "Defining a Calculation" on page 619

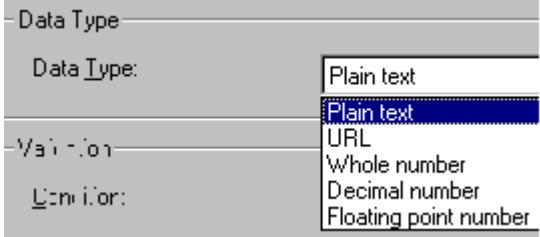
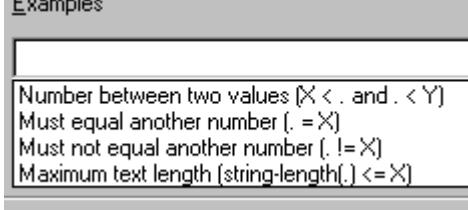
Defining an Error Message

Use the <errormessage localeRef="id"> element to define a message to display when the data is not valid. For example, "Must be a number between 1 and 10, inclusive."

The text may be within the <errormessage> tags or referenced using localeRef into the localeNNNN.xml file.

Defining Custom Validation

Use the <custom> element to change the standard values that appear in the following fields of the Custom Validation dialog.

Field	Screen image
Data Type	
Examples	

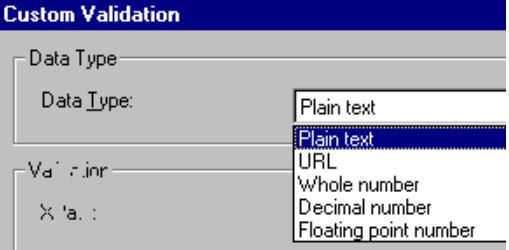
To change the list of options, modify the <custom> element of the configdatadesign.xml file. This section describes the <custom> element's attributes and child elements.

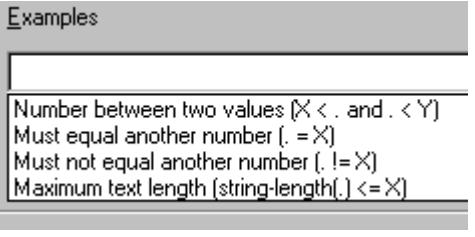
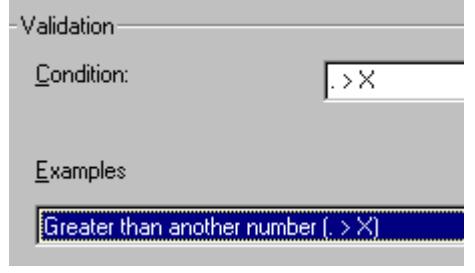
See Also: "selections" on page 352, "listchoice" on page 345

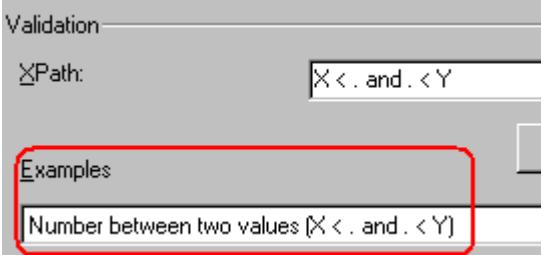
```
<datadesign>
  <validation>
    :
    <custom> (optional tag)
      <caption>
        <selections name="datatype">
          <listchoice> (0 or more)
```

```
<selections name="examples">
    <listchoice> (0 or more)
```

Element or Attribute	Description
<custom>	Specifies basic data types available when customizing validation. The types are defined in the <listchoice> elements (see below).
<custom visible="true false"	Controls whether the Custom Validation field appears on the Properties dialog.
<custom enabled="true false"	Controls whether the Custom Validation field is active or "grayed out" on the Properties dialog. If set to true , the Custom Validation field is active; if false , it is grayed out.
<caption localeRef="id">	Specifies text to display in the validation drop-down list. The default caption is " (Custom) ".
Selections element for Data Type field See Also: " "selections" on page 352	
<selections name="datatype">	The name must be datatype.
<selections enabled="true false"	Controls whether the drop-down list is active or "grayed out" on the Properties dialog.
<selections visible="true false"	Controls whether the drop-down list appears on the Properties dialog.
Listchoice element for Data Type field See Also: " "listchoice" on page 345	
<listchoice>	Defines the values in the Custom Validation dialog Data Type drop-down list.

Element or Attribute	Description
<listchoice value="simple-data-type">	<p>Enter each data type that should appear in the Custom Validation dialog's Data Type drop-down list.</p>  <p>The simple data types are defined by W3C XML Schema definition language 1.0. <i>See Also:</i> "Defining a Schema Fragment" on page 618</p>
<listchoice treeImg="id">	<p>Specifies the icon to display for this field in the Select Field or Group dialog. See "Icons on the Select Field or Group Screen" on page 626.</p> <p>Unlike toolbar icons, you cannot create your own.</p>
<listchoice localeRef="id">	<p>The text that describes this Data Type on the Custom Validation dialog.</p> <p>This element can refer to a string in the localeNNNN.xml file. Or, you can enter the string between the <listchoice> tags.</p>
<listchoice default="true false">	<p>Use this attribute to indicate the default choice in the Custom Validation dialog's Data Type drop-down list.</p>
Selections element for Examples field <i>See Also:</i> "selections" on page 352	
<selections name="examples">	The name must be "examples".

Element or Attribute	Description
<selections enabled="true false">	<p>Determines if the Examples drop-down list and label are active or “grayed out” in the Custom Validation dialog.</p> 
<selections visible="true false">	<p>Determines if the Examples drop-down list and label appear.</p>
Listchoice Element for Examples field See Also: "listchoice" on page 345	
<listchoice>	<p>Defines the values in the Examples drop-down list.</p>
<listchoice value="xpath-expression-example">	<p>The XPath expression appears in the Examples drop-down list of the Custom Validation dialog.</p> 

Element or Attribute	Description
<listchoice localeRef="id" -	<p>The text that describes the examples on the Custom Validation dialog.</p>  <p>This element can refer to a string in the localeNNNN.xml file. Or, you can enter the string between the <listchoice> tags.</p>

Saving Invalid Documents

Use the `publishinvalid` attribute of the `<standard>` element to determine if an invalid document should be saved. This attribute lets you decide if a user can save an XML document when the data in one or more fields does not satisfy the validation criteria.

In Data Entry mode, an example would be if a field requires a non-negative whole number, but the user does not insert a value in that field.

If the `publishinvalid` attribute's value is `true`, content is not checked for validity when it is saved.

If `false`, the content is checked for validity during a save. If the content is valid, it is saved. If it is invalid, the user is notified, and a custom script can be created to allow the content to be saved or prevent it.

NOTE If the `publishinvalid` attribute is not defined in the `<standard>` element, the default is `true` so that **eWebEditPro+XML** is backwards compatible with previous releases.

Custom Script that Handles Saving Invalid Files

To determine whether or not an invalid XML document should be saved, write a client-side script that is called when invalid content is found. The routine should ask if the user wants to save the invalid content. The script should return `True` to save the content, or `False` to abort the save.

Below is an example of such a script.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.instances[ "MyEditor1" ].onerror = myErrorHandler;

function myErrorHandler()
```

```
{  
    if (EWEP_STATUS_INVALID == this.status && "save" == this.event.source)  
    {  
        var strMsg = "Content is invalid.";  
        strMsg += "\nError Code: " + this.event.reason;  
        strMsg += "\nError Reason: " + this.event.message;  
        alert(strMsg);  
        return false; // prevent save  
    }  
}  
//-->  
</script>
```

The onerror Event

If the content is invalid, an onerror event is generated. (See Also: ["Event: onerror" on page 190](#)) This event provides two additional properties for the event object when it fires:

- reason (a numeric error code)
- message (text describing the error)

As with the regular onerror event, the `source` property is available. You can display the values of these properties in an error message that informs the user why the document is not valid.

NOTE You can localize the `invalidContent` message. See ["Modifying the Language of eWebEditPro+XML" on page 265](#).

Calculated Fields

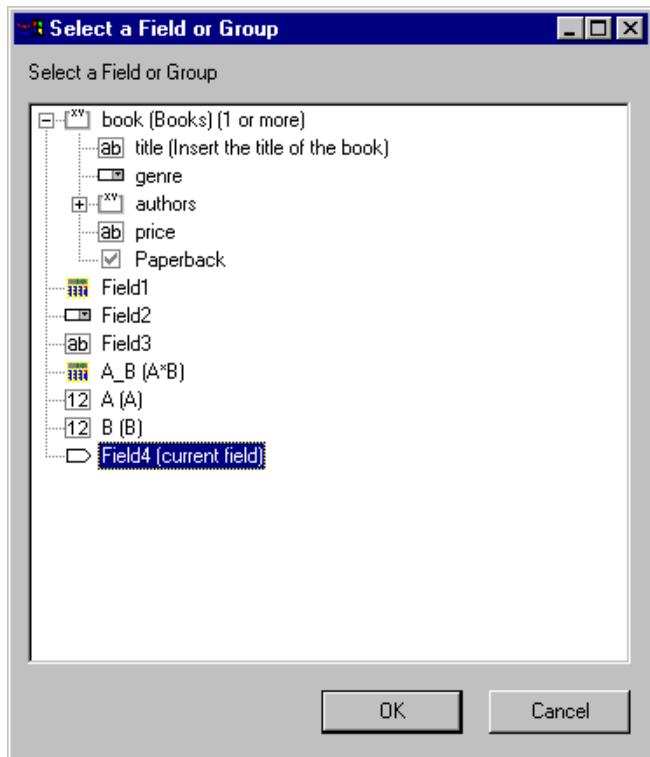
A Calculated Field lets you include values that are calculated, typically from values in other fields. For example, you can multiply two field values. The equation used is an XPath expression.

You can validate a Calculated Field, just like the Plain Text Field. For instance, the total of a series of numbers should be less than 100.

For more information about calculated fields, see the [eWebEditPro+XML User Guide's Chapter "Using the Data Designer."](#)

Icons on the Select Field or Group Screen

Use the following icons to denote a field on the Select A Field or Group Screen (illustrated below). These are the valid values for the `treeImg` attribute.



Icon	Name
	calculation
	calendar
	checkbox
	droplist
	fieldset
	hidden
	hyperlink

Icon	Name
	number
	password
	picture
	richarea
	text
	textbox

Learn More about XPath

Xpath Operators

Content copied from www.w3schools.com/xpath.

Operator	Description	Example	Result
Numerical expressions perform arithmetic operations on numbers. XPath converts each operand to a number before performing an arithmetic expression.			
+	addition	$6 + 4$	10
-	subtraction	$6 - 4$	2
*	multiplication	$6 * 4$	24
div	division	$8 \text{ div } 4$	2
mod	division remainder	$5 \text{ mod } 2$	1
Equality/ greater/ less than expressions test equality between two values			
=	equals	price = 9.80	true (if price is 9.80)

Operator	Description	Example	Result
!=	is not equal	price! = 9.80	false
<	less than	price < 9.80	false (if price is 9.80)
<=	less than or equal to	price <= 9.80	true
>	greater than	price > 9.80	false
>=	greater than or equal to	price >= 9.80	true
Boolean expressions compare two values			
or	or	price = 9.80 or price = 9.70	true (if price is 9.80)
and	and	price <= 9.80 and price = 9.70	false
Identifying a path and element			
.	the current element	. > 100	true if element exceeds 100
..	the current element's parent	count(../*)	counts the number of elements at the same level as the current element
Grouping and separating			
[]	predicate	../*[0]	../*[0] - returns the parent's first child element
	specify multiple elements	sum(X Y Z)	If X=1 and Y =2 and Z=3, sum(X Y Z) = 6

Xpath Functions

Content copied from www.w3schools.com/xpath.

Operator	Description	Syntax	Example	
last	Returns the position number of the last node in the processed node list	number=last()		
position	Returns the position in the node list of the node that is currently being processed	number=position()		
count	Returns the number of nodes in a node-set	number=count(node-set)		
name	Returns the name of a node	string=name(node)		
string	Converts the value argument to a string	string(value)	string(314) Result: '314'	
concat	Returns the concatenation of all its arguments	string=concat(val1, val2, ..)	concat('The', ',', 'XML') Result: 'The XML'	
starts-with	Returns true if the first string starts with the second string. Otherwise, it returns false.	bool=starts-with(string,substr)	starts-with('XML','X') Result: true	
contains	Returns true if the second string is contained within the first string. Otherwise, it returns false.	bool=contains(val, substr)	contains('XML','X') Result: true	
substring-after	Returns the part of the string in the string argument that occurs after the substring in the substr argument	string=substring-after(string,substr)	substring-after('12/10','/')	Result: '10'
substring-before	Returns the part of the string in the string argument that occurs before the substring in the substr argument	string=substring-before(string,substr)	substring-before('12/10','/')	Result: '12'
substring	Returns a part of the string in the string argument	string=substring(string,start,length)	substring('Beatles',1,4) Result: 'Beat'	
string-length	Returns the number of characters in a string	number=string-length(string)	string-length('Beatles') Result: 7	

Operator	Description	Syntax	Example
normalize-space	Returns the whitespace-normalized version of a passed string. All leading and trailing whitespace is stripped, and all sequences of whitespace get combined to one single space.	normalize-space('string')	normalize-space('some text') would return some text
translate	normalize	string=translate(value,string1,string2)	<ul style="list-style-type: none"> ● translate ('12:30','30','45') Result: '12:45' ● translate ('12:30','03','54') Result: '12:45' ● translate ('12:30','0123','abcd') Result: 'bc:da'
boolean	Converts the value argument to Boolean and returns true or false	bool=boolean(value)	
not	Returns true if the condition argument is false, and false if the condition argument is true	bool=not(condition)	not(false())
true	Returns true	true()	number(true()) Result: 1
false	Returns false	false()	number(false()) Result: 0
lang	Returns true if the language argument matches the language of the xsl:lang element. Otherwise, it returns false.	bool=lang(language)	
number	Converts the value argument to a number	number=number(value)	number('100') Result: 100
sum	Returns the total value of a set of numeric values in a node-set	number=sum(node-set)	sum(/cd/price)
floor	Returns the largest integer that is not greater than the number argument	number=floor(number)	floor(3.14) Result: 3

Operator	Description	Syntax	Example
ceiling	Returns the smallest integer that is not less than the number argument	number=ceiling(number)	ceiling(3.14) Result: 4
round	Rounds the number argument to the nearest integer	integer=round(number)	round(3.14) Result: 3

XPath References

To learn more about XPath, check these Web pages.

Topic	URL
W3C Spec - XPath 1.0 spec	http://www.w3.org/TR/xpath
Tutorial	www.w3schools.com/xpath/default.asp
Quick reference card	www.mulberrytech.com/quickref/XSLTquickref.pdf
Microsoft XPath reference for MSXML 4.0	xpath_ref_overview_0pph.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm>xpath_ref_overview_0pph.asp

Methods, Properties, and Events that Provide Access to Data Designer Fields

This section lists methods, properties, and events that you can use to access Data Designer fields. For an example of how to use some of these elements, see “Letting Users Execute a Custom Function on a Data Design Screen” on page 634.

Method, Property or Event	How used with Data Designer	Standard description on page
OuterXML	Returns the XML of the custom tag as a string, for example, <mytag>some text</mytag>.	109
Content	Retrieves or sets the content within the tag.	143

Method, Property or Event	How used with Data Designer	Standard descripti on on page
DataStyle	<p>Affects the control in data entry mode <i>and</i> text in presentation.</p> <p>Use this property to determine the appearance of the following field types in design and data entry mode: checkbox, plain text, select list, and calculated. For plain text, select list and calculated fields, the following style sheet (CSS) properties apply to the default presentation XSLT.</p> <ul style="list-style-type: none"> ● color ● all properties that start with background, font, text, and word. <p>The most important CSS properties are ones a designer can set in the user interface: font-weight, font-style, text-decoration, font-size, font-family, color, and background-color.</p>	143
Description	Read only	144
TagStyle	Same as DataStyle	145
TagName	Read only	145
TagType	Read-only. The value is always -1 for data designer fields.	145
IsDataField	Returns true if current tag is a Data Design tag.	96
GetXPath	Returns XPath of selected Data Design field.	93
FindDataField	Finds CXML data object specified by given xpath.	81
clicktag	<p>An event that occurs when a user clicks one of these fields types on a Data Design screen.</p> <ul style="list-style-type: none"> ● checkbox ● plain text ● select list ● button data fields in data entry mode 	182

Letting Users Execute a Custom Function on a Data Design Screen

This section explains how to let a user execute a custom command on any Data Design field. The API elements that provide these capabilities are listed in “[Methods, Properties, and Events that Provide Access to Data Designer Fields](#)” on page 632.

One example of a custom command is a button that, when clicked, displays a popup window. From there, the user can select from a database (such as a zip code listing), enter event-based information, or perform other activities which may be needed in the context of completing the Data Design screen.

A sample function installed with **eWebEditPro+XML** illustrates how a user can insert a button onto a Data Design screen. The button can perform any function that you assign to it. The sample button retrieves and displays the value of a field whose XPath is `/root/Field1`. To determine the button’s action, modify the sample function.

By default, the button appears on a drop-down menu on the Data Design toolbar in data entry mode (see illustration below).



The Sample Function

Define your buttons in the `customevents.js` file or on the page. The `ewebeditproevents.js` file includes an example of a button named `mybtn` (below). Notice that `mybtn` is defined in two places.

```

function eWebEditProInsertButton(sEditorName, name, caption, attributes)
{
    var sHTML = "<button name=" + toLiteral(name)+ " " + attributes + ">" + caption + "</
button>";
    eWebEditPro.instances[sEditorName].asyncCallMethod("pasteHTML", [sHTML], null, new
Function());
}

eWebEditProExecCommandHandlers["mybtn"] = function(sEditorName, strCmdName,
strTextData, lData)
{
    eWebEditProInsertButton(sEditorName, strCmdName, "Button", "");
}

eWebEditProExecCommandHandlers["clicktag"] = function(sEditorName, strCmdName, strTextData,
lData)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objXmlTag = objXmlDoc.ActiveTag();
    if((typeof objXmlTag != "undefined") && (objXmlTag != null))
    {
        var sXPath = objXmlTag.GetXPath();
        alert("Clicked button: " + strTextData + " XPath: " + sXPath);
        if ("mybtn" == strTextData)
        {
            sXPath += "/Field1";
            var objDataFld = objXmlDoc.FindDataField(sXPath);
            if (objDataFld)
            {
                var sValue = objDataFld.getPropertyString("Content");
                alert(sValue);
            }
            else
            {
                alert("Could not find field with XPath: " + sXPath);
            }
        }
    }
}
}

```

The clicktag handler provides the button's action. The action can be anything that you wish, such as setting the value of another field, searching a database, or generating a popup window. In this example, the button attempts to read the value of Field1.

Defining the Button

You define Data Design buttons in the configuration XML. mybtn is an example in configdatadesign.xml. The list appears as a drop-down menu on the Data Design toolbar.

```

<command name="cmddsgbutton" style="list" maxwidth="12">
    <image key="bbtn"/>
    <caption localeRef="cmdBtn"/>
    <tooltiptext localeRef="cmdBtn"/>
    <selections name="commands">
        <listchoice command="" localeRef="sSelBtn"/>

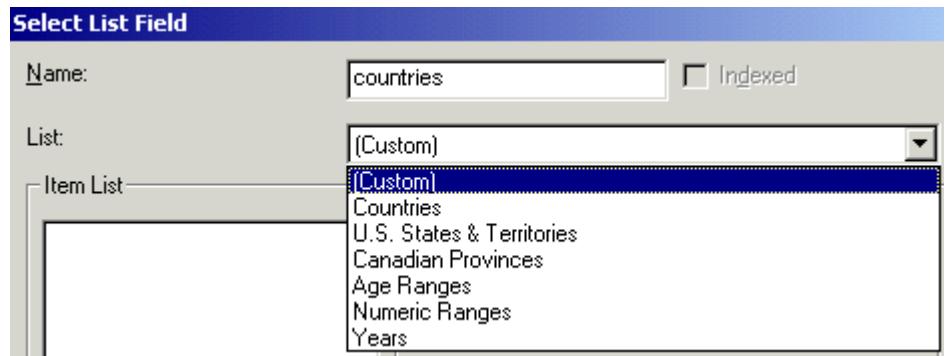
```

```
<!-- remove the following listchoice and add your own -->
<listchoice command="mybtn" localeRef="cmdBtn"/>
</selections>
</command>
```

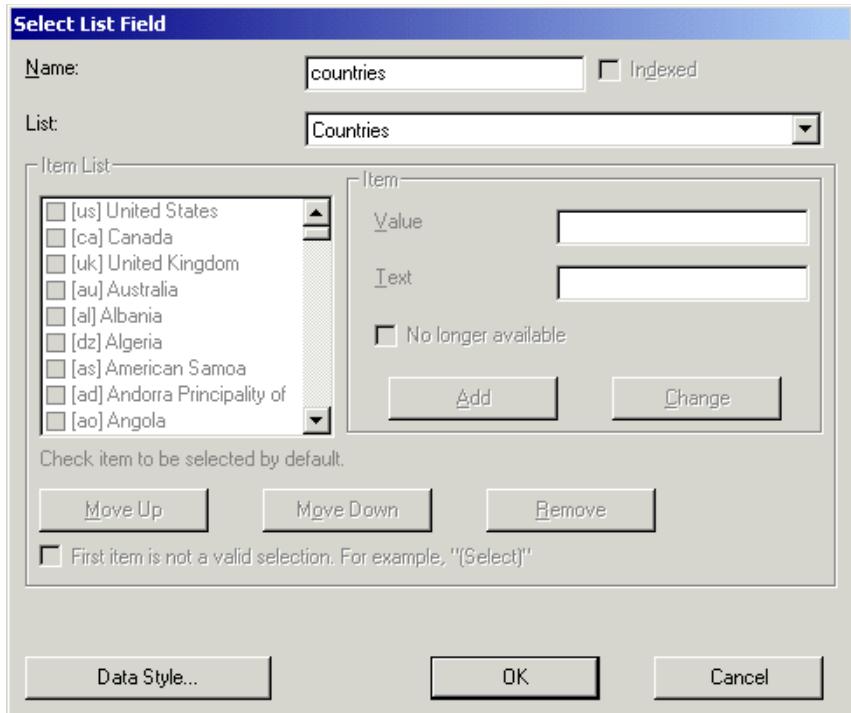
NOTE You should either remove the cmddsgbutton command from the configuration or remove "mybtn" and implement your own buttons.

Specifying Items in a Select List Field

This section explains how to specify an XML document of items to populate the Item List field on the Select List Field dialog.



If you choose an entry from the **List** field dropdown (such as **Countries**), the XML document populates the **Item List** field (as illustrated below).



This section explains this feature via the following subtopics.

- “[Advantages of Storing List Items Externally](#)” on page 637
- “[Specifying an External List](#)” on page 638
- “[New API Methods in XML Data Object](#)” on page 640
- “[Examples of Creating Select Lists](#)” on page 640

Advantages of Storing List Items Externally

An external list can reside in the configuration data, an external document (such as `[eWebEditProPath]/countries.xml`), or in a database that is queried from a dynamic web page, such as ASP. In fact, the list could be authored within **eWebEditPro+XML**. (For an example of specifying a list within the configuration data, see “[Item Element Attributes](#)” on page 639.)

Although you can type any group of items into the Select List Field dialog, this feature lets you reference an xml-based, external document of potentially hundreds of items (such as United States zip codes). Referencing such a document saves you the time of entering items into the dialog by hand. Also, the same external document can be referenced in several fields on the same screen or on several screens. If the list later needs updating, you only need to update it once, and all fields referring to it are automatically updated.

Specifying an External List

You can specify the items in an external list through the configuration or JavaScript. In Data Entry mode, the Presentation XSLT is not automatically updated. If out of date, it displays the raw XML data value rather than its associated caption.

The default data lists are provided within the `cmddsglistcontrol` command of the configuration data's `<datadesign>` element, as shown below. You can create your own lists using the defaults as a model.

```
<command name="cmddsglistcontrol">
  <image key="droplist"/>
  <caption localeRef="cmdListFld"/>
  <tooltiptext localeRef="cmdListFld"/>
  <selections name="datalists">
    <listchoice data="countries" localeRef="sCountries"/>
    <listchoice data="USPS-US" localeRef="sUSState"/>
    <listchoice data="USPS-CA" localeRef="sCaPrvnc"/>
    <listchoice data="ageRange" localeRef="sAgeRng"/>
    <listchoice data="numRange" localeRef="sNumRng"/>
    <listchoice data="years" localeRef="sYrs"/>
  </selections>
</command>
```

Each `<listchoice data="{data-list-name}">` item is associated with a `<datalist name="{data-list-name}">` within the `<dataentry>` element. Here is the structure.

```
<datalist name=~" src="{URL}" select="{XPath}" captionxpath="{XPath}" valuxpath="{XPath}"
namespaces=~" [validation="select-req"]>
  [<schema datatype="{basic-data-type}">{optional schema fragment}</schema>] (may optionally
  specify items in addition to those in the 'src'. Typically for a "(Select)" item.)
  <item value=~" [default="true"] [localeRef=~"]>(display text if no localeRef)</item>
  :
</datalist>
```

Note In data entry mode, only `<datalist>` elements are needed.

Datalist Element Attributes

Attribute	Description
src	URL to XML document used to create the list. For example [<code>eWebEditProPath</code>]/countries.xml
select	An XPath expression that identifies the element that contains the caption and value.
captionxpath	An XPath expression relative to the select attribute that specifies the caption value.

Attribute	Description
valuexpath	An XPath expression relative to the select attribute that specifies the value itself.
namespaces	Namespaces used in the XPath expressions
validation	Set to value <code>select-req</code> to identify the first item on the list as an invalid selection. An example of an invalid selection is instructional text such as Select a state , which is not an actual value.

Schema Element Attributes

Attribute	Description
datatype	Any basic schema datatype, for example, string, nonNegativeInteger. The default is string.

Item Element Attributes

Use the Item element to specify a list within the configuration data that is available as a choice for the **List** field of the Select List Field dialog. The following sample list, provided in the configdatadesign.xml file, exemplifies how to use this element.

```
<datalist name="ageRange" validation="select-req">
    <item value="" localeRef="sSel" />
    <item value="10">0-15</item>
    <item value="20">16-25</item>
    <item value="30">26-35</item>
    <item value="40">36-45</item>
    <item value="50">46-55</item>
    <item value="60">56-65</item>
    <item value="70">66-75</item>
    <item value="80">76-85</item>
    <item value="90">86+</item>
</datalist>
```

NOTE

If `<item>` elements are provided in addition to an `src` value within the `datalist` element, the item elements appear first, followed by values in the `src` document.

Attribute	Description
value	The value itself

Attribute	Description
default	true or false, indicating whether this value is the default selection
localeRef	Localization identifier. This value specifies the string in the locale file that translates the caption into a local language. See Also: "Modifying the Language of eWebEditPro+XML" on page 265
(text between the tags)	The display text, if localeRef is not used

New API Methods in XML Data Object

These methods apply only when a Select List Data Designer field is the active tag.

Method	Function
"Method: LoadList" on page 105	Creates list of items from an XML document
"Method: ClearList" on page 62	Removes all items from a list
"Method: AddListItem" on page 53	Adds item to end of list
"Method: RemoveListItem" on page 115	Removes item from list
"Method: ListLength" on page 104	Returns number of items in list

Examples of Creating Select Lists

Employee List

XML Source Document

From Select/Option created in editor's design page, using the design page itself as source.

```
<select id="employees" onblur="design_validate_select(this)" ektdesignns_nodetype="element"
ektdesignns_validation="select-req" ektdesignns_name="employees">
  <option value="(Choose an employee)" selected="selected">(Choose an employee)</option>
  <option value="Anthony">Anthony</option>
  <option value="Burt">Burt</option>
  <option value="Charles">Charles</option>
  <option value="Danielle">Danielle</option>
</select>
```

Configuration Data

```
<datalist name="employees" src="/mysite/designpages.asp?id=employees" select="/select/option"
captionxpath=". " valuemxpath="@value" validation="select-req"/>
```

JavaScript

```
objDDField.LoadList( "/mysite/designpages.asp?id=employees", "/select/option", ". ", "@value",
" " );
```

Remarks

In a CMS, data designers can create their own dynamic lists.

Nested Group Boxes Created in Editor and Using XML Document

XML Source Document

```
<root>
    <employees>
        <employee>
            <name>Anthony</name>
            <ext>123</ext>
        </employee>
        <employee>
            <name>Burt</name>
            <ext>124</ext>
        </employee>
        <employee>
            <name>Charles</name>
            <ext>125</ext>
        </employee>
        <employee>
            <name>Danielle</name>
            <ext>126</ext>
        </employee>
    </employees>
</root>
```

Configuration Data

```
<datalist name="employees" src="/mysite/data.asp?id=employees"
select="/root/employees/employee" captionxpath="name" valuemxpath="name"/>
```

JavaScript

```
objDDField.LoadList( "/mysite/data.asp?id=employees", "/root/employees/employee", "name",
"name", " " );
```

Remarks

In a CMS, data entry authors can create their own dynamic lists.

A W3C XML Schema

This example demonstrates how to reference an external list of U.S. state abbreviations. This list is maintained on <http://www.irs.gov/eforms>.

XML Source Document

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

```

<xsd:schema targetNamespace="http://www.irs.gov/eforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.irs.gov/eforms" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:annotation>
        <xsd:documentation>
            <Description>IRS 887x Schema - Base types commonly used across schema
files</Description>
            <Version>Version 2.2</Version>
            <ReleaseDate>April 15, 2004</ReleaseDate>
        </xsd:documentation>
    </xsd:annotation>
    :
    <xsd:simpleType name="StateType">
        <xsd:annotation>
            <xsd:documentation>State abbreviations, a.k.a. state
codes</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="AL">
                <xsd:annotation>
                    <xsd:documentation>Alabama</xsd:documentation>
                </xsd:annotation>
            </xsd:enumeration>
            <xsd:enumeration value="AK">
                <xsd:annotation>
                    <xsd:documentation>Alaska</xsd:documentation>
                </xsd:annotation>
            </xsd:enumeration>
        <:
    
```

Configuration Data

```

<datalist name="USPSStates" src="8871and8872commonDefV22.xsd"
select="/xsd:schema/xsd:simpleType[@name='StateType']/xsd:restriction/xsd:enumeration"
captionxpath="xsd:annotation/xsd:documentation"
valuexpath="@value"
namespaces="xmlns:xsd='http://www.w3.org/2001/XMLSchema' "
validation="select-req">
    <item value="">(Select a state)</item>
</datalist>

```

JavaScript

```

objDDField.LoadList("8871and8872commonDefV22.xsd",
"/xsd:schema/xsd:simpleType[@name='StateType']/xsd:restriction/xsd:enumeration",
"xsd:annotation/xsd:documentation", "@value", 'xmlns:xsd="http://www.w3.org/2001/XMLSchema"' );

```

Remarks

WARNING!

Because the XPath expressions use the 'xsd:' namespace prefix, the namespace declaration must be given. If the schema namespace is not specified, the list is empty.

US Postal Service State Abbreviations

This example demonstrates how to reference an external list of U.S. postal codes. This list is maintained on <http://www.daml.org/2003/02/usps/state.owl>.

XML Source Document

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
  xmlns      = "http://www.daml.org/2003/02/usps/usps-ont.owl#"
>

<rdf:Description rdf:about="">
  <rdfs:comment>US Postal Service State Abbreviations</rdfs:comment>
</rdf:Description>

<State rdf:ID="AL">
  <rdfs:label>ALABAMA</rdfs:label>
  <USPostalCode>AL</USPostalCode>
</State>

<State rdf:ID="AK">
  <rdfs:label>ALASKA</rdfs:label>
  <USPostalCode>AK</USPostalCode>
</State>
:
```

Configuration Data

```
<itemlist src="state.owl.xml" select="//usps:State"
  captionxpath="rdfs:label" valuexpath="usps:USPostalCode"
  namespaces='xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:usps="http://www.daml.org/2003/02/usps/usps-ont.owl#"'/>
```

JavaScript

```
objDDField.LoadList("state.owl.xml", "//usps:State", "rdfs:label",
  "usps:USPostalCode", 'xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:usps="http://www.daml.org/2003/02/usps/usps-ont.owl#"');
```

Remarks

WARNING!

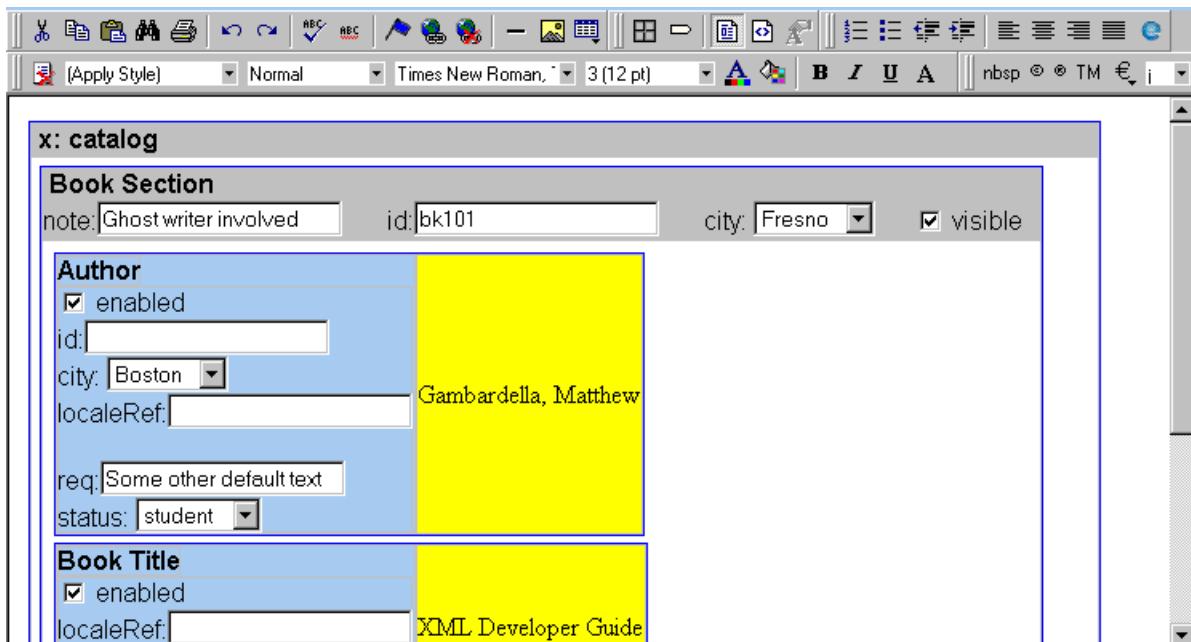
Because the document declares a default namespace (<http://www.daml.org/2003/02/usps/usps-ont.owl#>), the XPath expressions must specify a namespace prefix. In this example, `usps` was used. If the prefix to `State` and `USPostalCode` were not added, the list would be empty.

Original XML Functionality

Entering XML Content

eWebEditPro+XML is a WYSIWYG editor that accepts user input and tags that input as XML elements and attributes. Because the page appears in a Web browser, contributors do not need to know XML, nor do they know that their content is being saved as XML. The XML tags are merely custom tags within the XHTML content.

The following graphic illustrates an **eWebEditPro+XML** screen formatted to collect XML elements and attributes.



While the user is entering XML data, the **eWebEditPro+XML** toolbar is available to format text. The resulting Web content can contain XML and HTML tags.

NOTE

If you have any problems when you begin using XML, please see "Troubleshooting Problems with eWebEditPro+XML" on page 655. This section explains how to solve problem that may occur when users begin working with **eWebEditPro+XML**.

Enabling Entry of XML Content

When loading, the editor imports XML information that determines the presentation of the custom elements and attributes. A user then interacts with

the editor to insert and modify XML content. When a user saves the content, the XML tags and content are preserved.

When the editor loads a full XML document, functionality such as `getDocumentHTML` and `getHeaderHTML` operate with the XML information rather than XHTML information. The document and header retrieved will be a XML document and header, and XML header information is preserved. Also, when a script deals with XML data, the editor maintains this information and uses it to display content to the user. A script can load XML, retrieve XML, validate and store the data.

The developer can also use standard menu functionality to create custom buttons and dropdown lists that let a user insert XML content. JavaScript API properties, methods, and events designed for XML tagging let developers extend and add their own features.

Retrieving XML Content

You retrieve custom tags to **eWebEditPro+XML** the same way that you retrieve HTML content. Here is an example:

```
strValue = eWebEditPro.instances.MyContent1.editor.getDocument()
```

Also, because the data is tagged as XML, you can display it using standard XML retrieval techniques. For example, you can display XML content to any XML-capable output device by using an Extensible Style Language Transformation (XSLT) file.

Validating XML Content

eWebEditPro+XML provides two levels of validation:

- Validation of items selected by the user, so that only valid elements, attributes, and values can be selected while editing. This form of validation prevents the user from inserting invalid content.
You must use a schema to perform this kind of validation.
- Validation of the full XML document in one all-encompassing check. This is done by validating the entire content against a schema or a DTD. (The schema or DTD must be specified in the header of the XML.)

This form of validation is done externally. The full content is extracted and placed into the validation method. Often, the user is then notified of any problems. You can use the error properties of the XML Object Interface to identify problems with validation. See Also: "["XML Object Interface" on page 719](#)

Next, you can optionally run a transformation on the validated content to correct any formatting problems. The result is then placed back into the validation method.

Alternatively, you can register a transformation file that automatically transforms content when it is extracted. The content can then be sent into the validation method.

You can base your XML structure on a standard schema or DTD, such as NewsML, or you can create your own schema or DTD.

See Also: "["Validating XML Content" on page 663](#)

Saving XML Content as a Whole XML Document

If the root tag is a custom XML tag, **eWebEditPro+XML** saves content out of the editor as a whole XML document.

If the root tag is XHTML with XML tags and you want to save content out of the editor as a whole XML document., set the editorGetMethod to `getDocument` instead of `getBodyHTML`. You can set the editorGetMethod value in `ewebeditprodefaults.js` or on the Web page *prior* to creating the editor, as illustrated below.

```
eWebEditPro.parameters.editorGetMethod = "getDocument";  
// code to create the editor goes here
```

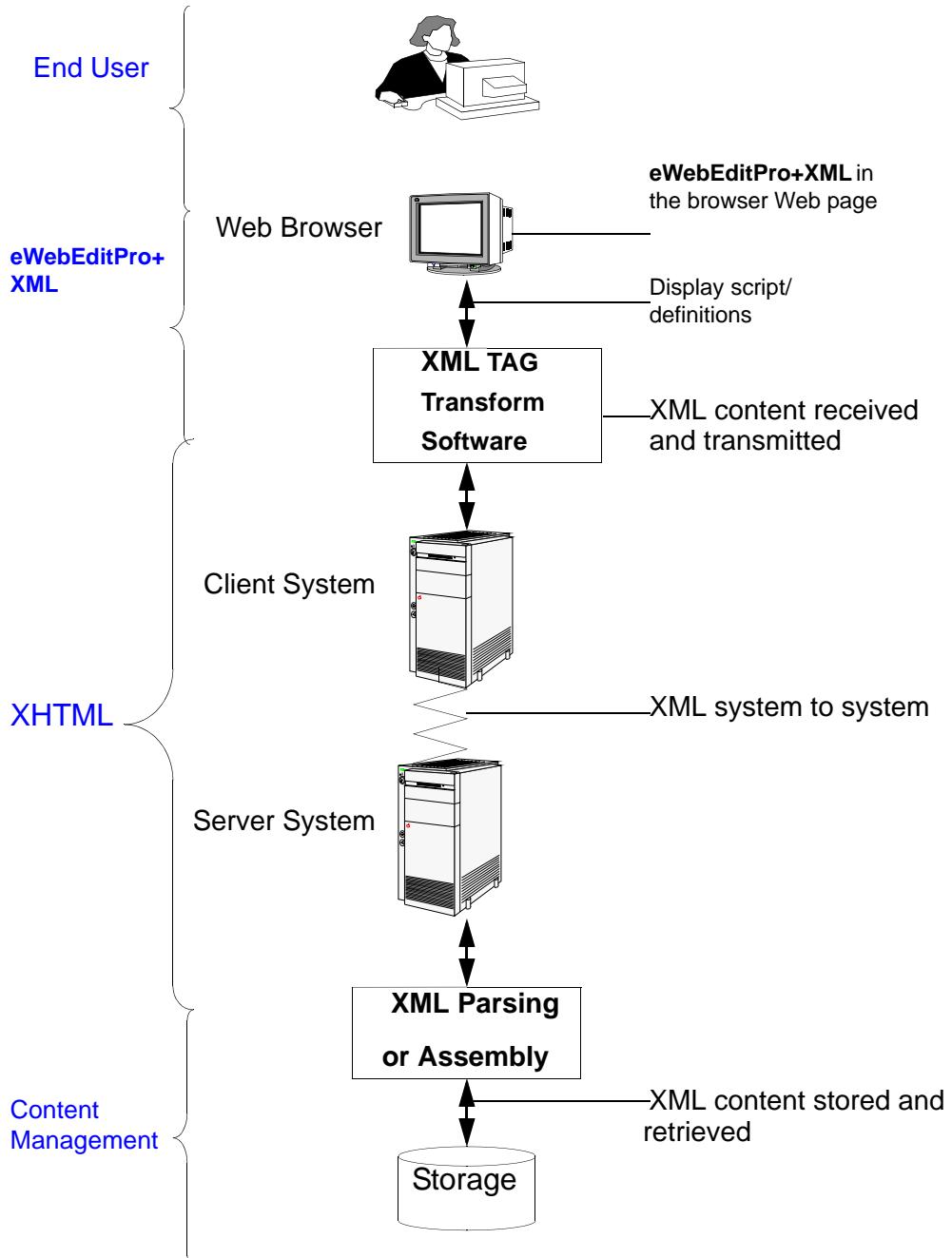
For details on showing the root XML element, see "Docxml Element" on [page 696](#).

For details on editorGetMethod, see "[Property: editorGetMethod](#)" on page 182.

How the Editor Stores and Retrieves XML Content

The following steps explain how **eWebEditPro+XML** stores and retrieves XML content. The accompanying graphic illustrates these steps.

1. XML content is retrieved from a content management system.
2. Content is loaded into **eWebEditPro+XML**.
3. The user interacts with the editor, inserting or modifying content.
4. XML content is sent back to the content management system.
5. The content management system saves the edits.



Sample Files

Several sample files that demonstrate how to implement the XML feature are installed with **eWebEditPro+XML**. Studying these files and the source files used to render them can help you carry out your own implementation.

To view the sample files, click the Windows **Start** button, then follow this path:

Programs > Ektron >ewebeditpro5 > Samples > xml

The source files used to create the sample files are located in
`ewebeditpropath\samples\xml`.

Where to Define the Appearance of XML Tags

There are three ways to define the appearance of custom XML element and attribute tags within the editor. You can

- Define tags in config.xml.
- Assign the external XML file (`xmlinfo.xml`) to the `xmlInfo` attribute in the `eWebEditPro.parameters` object. This is done during the initialization of the editor.
- Assign the `xmlinfo` file to the `XmllInfo` property in the **eWebEditPro+XML** editor. This is done while the user is editing content.

If more than one of these definitions is in place, the editor uses the “closest” definition. In the list above, 3 is the closest, followed by 2 and 1.

NOTE	While the XML information describes a tag's appearance, a schema describes the structure of the elements and attributes. For each attribute, the schema also describes characteristics, such as whether it is required, a default value, and response type (generic text, a set of values, or a Boolean value).
-------------	---

Defining Tags in Config XML

You can define the appearance of elements and attributes in the **features > customtag** feature of the config.xml data. For a description of the `customtag` elements, see “[The Custom XML Tag DTD and Schema](#)” on page 678.

An advantage of this approach is that it keeps all definitions of editor functionality in one file.

When you install **eWebEditPro+XML**, the config.xml file contains sample custom tag definitions that you can reference when designing your own tags. Here is the beginning of that section of the file.

```
<!-- These are sample tags. Your own tags must be defined here.
Attributes not included here will derive their values from the tagdefault defined above.-->
<tagspec name="mysubject" type="horizontal" visible="true"
          style="font-family:arial; font-weight:bold; background-color:#cccccc;
                  border:solid blue 1pt; margin:2px; width:95%;"
          dstyle="font-family:arial; font-weight:normal; background-color:white;
                  padding:2px; width:95%;"
          astyle="font-size:normal;font-weight:normal" ashow="true">
<!-- If a caption is not given, the name of the tag is used as the caption. -->
```

```

<caption>Subject:</caption>
<simtaglist name="deftaglist">
    <simtag name="myto">To:</simtag>
    <simtag name="mycc">Cc:</simtag>
</simtaglist>
</tagspec>

```

Assigning XML Data in a Script

There are three ways that you can dynamically assign XML and custom tag configuration data:

- the `xmlInfo` attribute
- the `XmlInfo` property
- the `cmddefinetags` command

Each is described below.

NOTE

Case is *very important* for the variables described in this section. For example, “`xmlInfo` attribute” has a lowercase X, lowercase L and uppercase I. “`XmlInfo` property” has an uppercase X, lowercase L and uppercase I. Changing the case of any letter can cause problems.

The `xmlInfo` Property

Use the `xmlInfo` property to dynamically assign XML and custom tag configuration data that is external to the normal configuration data. This property is the preferred way to assign this data.

You can use `xmlInfo` to change settings that involve tag definitions, schemas, and transformations of full XML documents. You *cannot* use it to change dialog or feature settings.

Description: Specifies an `xmlInfo` file or stream that contains information on defining custom tags, schemas, and transformations. The data must validate against the `xmlinfo.dtd` or `xmlinfo.xsd` definition files.

Normally, this is accessed after the ready notification but can be used whenever the editor is operational. It is used by the core JavaScript to load data during initialization.

When a new `xmlInfo` file loads successfully, a clean operation is performed to reformat the content according to the new definition. If the loading fails, there is no automatic cleaning.

Type: String

You define access to `xmlInfo` the same way that you define access to the `config` attribute within the core JavaScript. Below is an example of how to use it.

See Also: “[“`xmlInfo`” on page 708](#)

```

<input type="hidden" name="MyContent1" value="">
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{

```

```

eWebEditPro.parameters.xmlInfo = "debugtags.xml";
eWebEditPro.create("MyContent1", "100%", 400);
}
//-->
</script>

```

This function sets `xmlInfo` definition data into the editor during run time.

```

function SetNewXmlInfo(sEditorName, sXmlInfo)
{
    var objInstance = eWebEditPro.instances[sEditorName];
    objInstance.editor.setProperty("XmlInfo", sXmlInfo);
}

```

Assigning the `XmlInfo` Property

You can dynamically assign configuration data in the `xmlinfo` file through the `XmlInfo` property. For more information, see ["Property: XmlInfo" on page 709](#).

The `cmddefinetags` Command

You can dynamically assign XML and custom tag configuration data external to the normal configuration data using the `cmddefinetags` command. You can use this command whenever the editor is operational but not during initialization.

Command: `cmddefinetags`

Description: Instructs the `customtag` feature to read tag definitions from a tag definition file or an XML stream of tag definition data. The XML must conform to the `xmlinfo.xsd` or `xmlinfo.dtd`.

Text: Either the path to a `customtag.xml` file or XML custom tag data, which should be validated against the `xmlinfo.xsd` or `xmlinfo.dtd` file.

IData: If this value is 0, the older definitions of tags are replaced. If the value is 1, the tags are preserved. Any other values are reserved.

WARNING!

If the `IData` value is 0, all previous tag definitions are replaced with the definitions in this data.

Here is an example of how to use this function in JavaScript.

```

function DefineCustomTags(sEditorName)
{
// Send command to define the tags, and to replace any other tags.
eWebEditPro.instances[sEditorName].editor.ExecCommand("cmddefinetags", "poemtags.xml", 0);
}

```

Determining How a Tag Displays

You control a custom element tag's look by the way you define it in the configuration data. You can control the following items:

- tag visibility
- content availability
- element tag presentation

- element content field presentation
- attribute tag presentation
- glyph display

Here is an example of a tag definition.

```
<tagspec name='author' type='horizontal' visible='1' render='1' edit='1'
    style='column-span:10;font-weight:bold;background-color:lightblue; border:solid
        blue 1pt; font-family:arial; margin:3pt'
    dstyle='font-weight:normal;background-color:yellow; color: black; font-family:times
        new roman; margin:3pt'
    astyle='font-size:normal;font-weight:normal' ashow='true'>
    <caption>Author</caption>
    <simtaglist name='deftaglist'>
        <simtag name='title'>Book Title</simtag>
    </simtaglist>
    <glyph src='[eWebEditProPath]/btmhy.gif' visible='0' />
</tagspec>
```

To define attributes for an element, use a schema.

See Also: ["Working with Schemas" on page 666](#)

Note that a user can only move a custom tag when a glyph is defined for the tag. To move a tag in this case, the user selects and moves the glyph.

For more information on the tagspec element's style attributes, see ["Tagspec Element" on page 684](#) and ["Tagdefault Element" on page 693](#).

Tag and Data Sections

A key concept to understand is that an element tag has two sections, and each section can be formatted differently. The following graphic illustrates the sections.



Section	Description	Formatted using this attribute
Tag	Defines a field label that prompts the user for input. Note that the <code>caption</code> attribute stores text for the field label.	<code>style</code>
Data	Stores the content that the user enters	<code>dstyle</code>

You can also define an attribute tag's appearance using the `astyle` attribute.

See Also: ["Working with Attributes" on page 659](#)

Tag Definition Short Cut

If you define several tags that use the same display styles, use the shortcut tag, `simtag`, to define tags quickly. This tag lets you define formatting attributes shared by several other tags. This tag can reduce the effort required to define tags and the size of the XML data.

In the following example, the custom tag `title` inherits the attributes of the `author` tag.

```
<tagspec name='author' type='horizontal' visible='1' render='1' edit='1'
         style='column-span:10;font-weight:bold;background-color:lightblue; border:solid
                blue 1pt; font-family:arial; margin:3pt'
         dstyle='font-weight:normal;background-color:yellow; color: black; font-family:times
                new roman; margin:3pt'
         astyle='font-size:normal;font-weight:normal' ashow='true'>
<caption>Author</caption>
<simtaglist name='deftaglist'>
    <simtag name='title'>Book Title</simtag>
</simtaglist>
<glyph src='[eWebEditProPath]/btnhy.gif' visible='0' />
</tagspec>
```

For more information, see ["Simtaglist Element"](#) on page 694.

Inserting Custom Tags

How Users Insert Custom Element Tags

When a user inserts a custom element tag, he follows these steps.

1. Move the cursor to the element after which you want the new element to appear.
2. Right click the mouse. The context menu appears.
3. Click **Insert Custom tag**.
4. If any elements can be added at the current location, a list of valid elements appears.



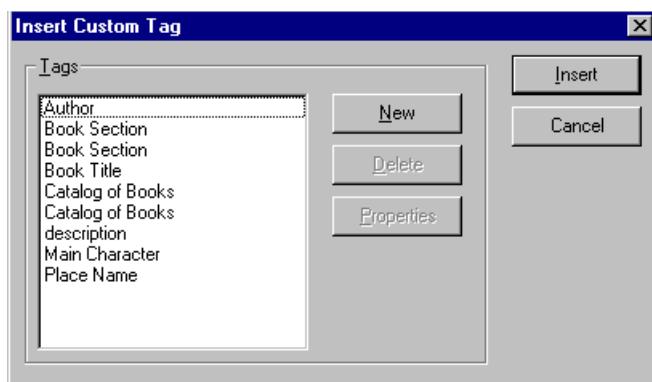
5. To insert any element, click it.
6. If the selected element has any attributes, the Custom Tag Attributes dialog appears. Required attributes appear in the left pane (**Used Attributes**) and cannot be removed.



Select any attribute in the right pane (**Offered Attributes**) that you want to insert. For each attribute in the left pane (**Used Attributes**), you can select a value.

Click OK to insert the element and attributes.

7. If no elements can be added at the current location or you click **Insert Custom tag** from the elements menu (illustrated in Step 4) and the developer lets the user insert any tag, the Insert Custom Tag dialog appears.



8. Select the tag to insert or click **New** to insert a new tag.
9. If you want to edit the tag's properties, click **Properties** and update the information on the custom tag properties screen. The tag's default properties are defined in the xml information under **features > customtag > tagdefinitions> tagdefault**.
10. Click **Insert** to insert the custom tag.

See Also: "How Users Insert Attribute Tags" on page 660

How Developers Insert Custom Tags

Although users can insert custom tags with the Insert Custom Tag dialog, many developers preload tags onto the Web page. Some developers disable the dialog so that users can insert attributes and XML content onto the page but cannot insert tags.

See Also: "Custom XML Dialog Boxes" on page 714

Inserting a custom element and attribute onto a Web page is identical to inserting any HTML fragment. To do so, use the `pasteHTML` method (see ["Method: pasteHTML" on page 109](#)). For example:

```
MyContent1.pasteHTML( "<mya attr='1'>hello</mya>" )
```

To load a Web page that displays three tags (summary, information and references), use this code:

```
eWebEditPro.instances[sEditorName].editor.pasteHTML( "<summary></summary><information></information><references></references>" )
```

As described in ["Where to Define the Appearance of XML Tags" on page 648](#), there are many ways to preload a Web page with XML content and allow a user to interact with it. The above example illustrates the simplest, least flexible method for doing so.

Most developers prefer an xml file that stores rich formatting information about elements. The xml file, in turn, refers to a schema that supplies attribute information and can validate the content.

Preloading Element Tags with Content

Using the `pasteHTML` method, you can preload any element tag with content to help the user respond. For example, if you enter the following code, the image below indicates the resulting Web page.

```
eWebEditPro.instances[sEditorName].editor.pasteHTML( "<summary>Enter summary here</summary><information></information><references></references>" )
```



Users replace **Enter summary here** with their own text.

If you use the `xmlInfo` attribute to dynamically assign XML and custom tag configuration data, you can define content for tags and attributes in that file. Here is an example from a sample file provided with **eWebEditPro+XML**.

```
<book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications with
    XML.</description>
</book>
```

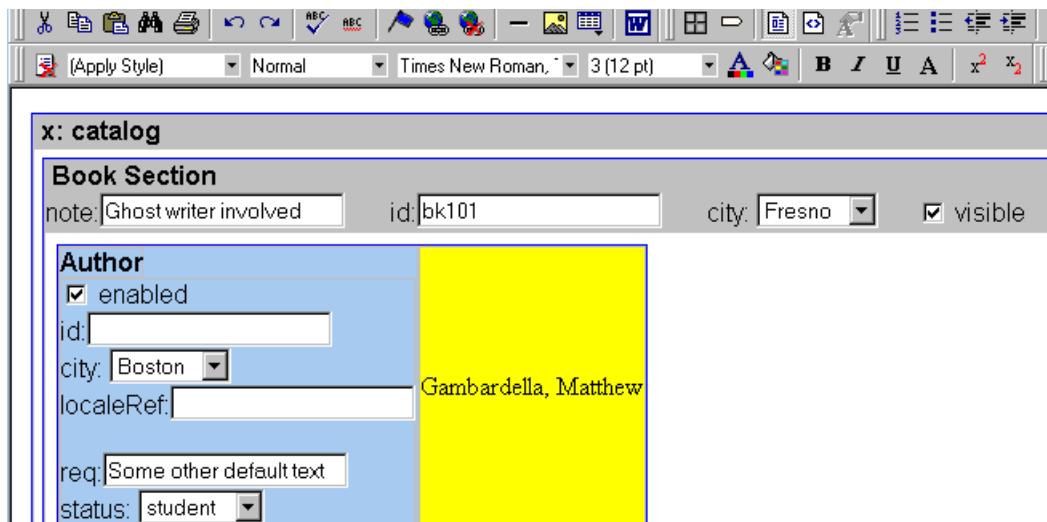
Troubleshooting Problems with eWebEditPro+XML

This section describes two common problems that may occur when working with **eWebEditPro+XML**, and explains how to solve them.

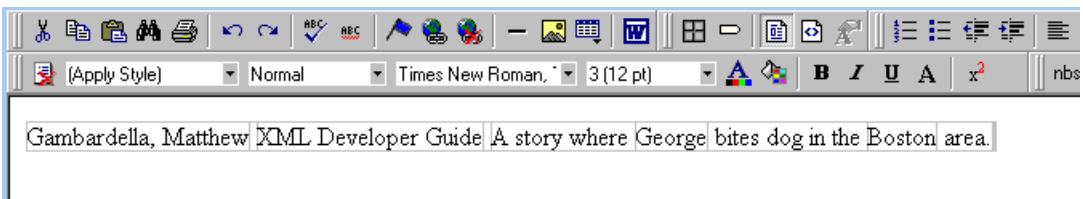
- No Formatting of XML Tags
- Attributes Do Not Display and Are Not Offered to User

No Formatting of XML Tags

This symptom is that the XML tags do not display according to formatting specifications. For example, you expect to see this:



but you see this instead

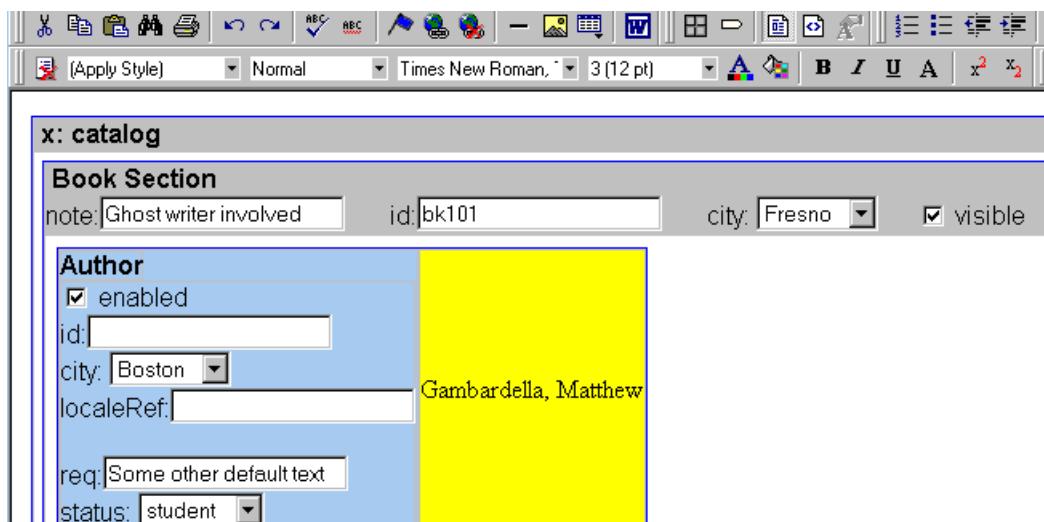


The possible causes of this problem are listed below.

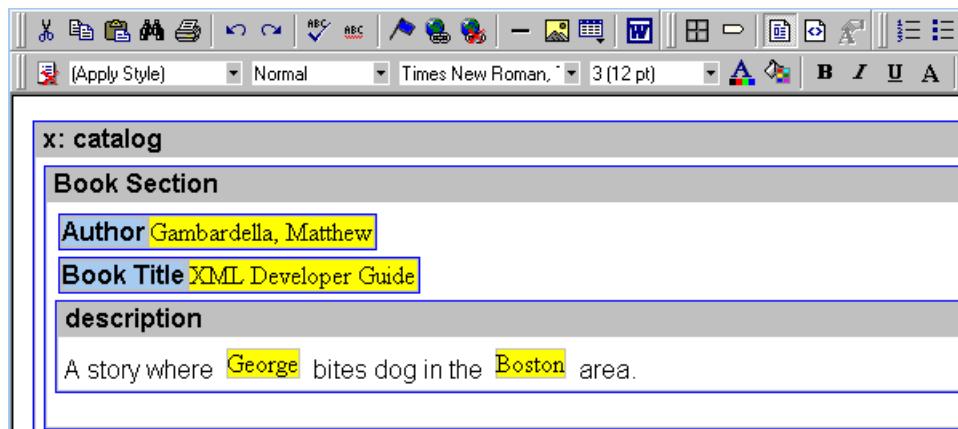
Possible Cause	Resolution
Invalid or missing license key	Review license key in ewebeditprolicense.txt file (in localhost/ewebeditpro5 directory). If key is missing or does not include the characters XML , obtain valid license key from Ektron and insert it into ewebeditprolicense.txt.
The XML configuration information does not include a <customtag> section. By default, the XML configuration information includes a <customtag> section. However, if you do not use the default configuration, and your XML configuration information does not include a <customtag> section, XML tags do not appear in the editor.	Add a <customtag> section to the XML configuration information. Add tag definition information to that section. See Also: "CustomTag Element" on page 683

Attributes Do Not Display and Are Not Offered to User

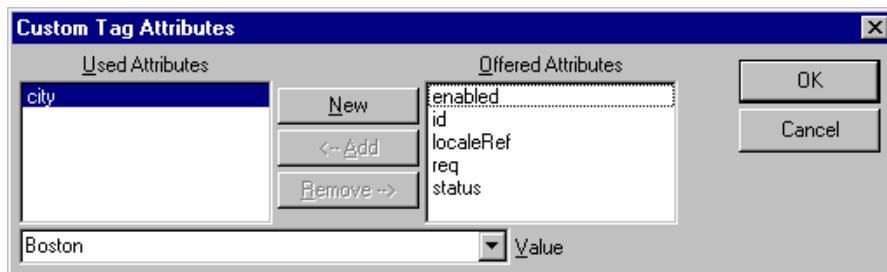
This symptom is that tag attributes do not display, nor are they offered to the user in the Custom Tag Attributes dialog. For example, you expect to see this:



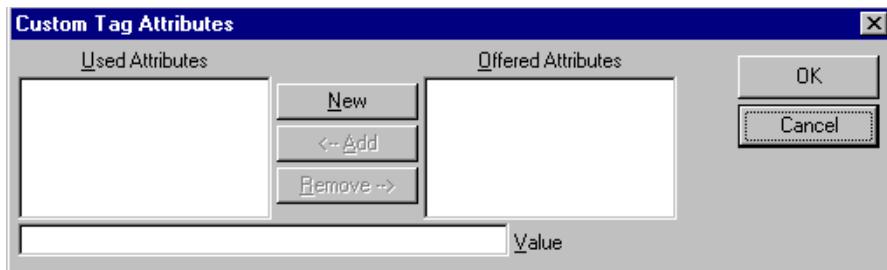
but you see this instead (note that the attributes do not appear).



Also, you expect available attributes to appear in the Custom Tag Attributes dialog, like this:



but none appear, like this



Possible Cause	Resolution
Microsoft XML parser was not installed during the eWebEditPro+XML installation.	<p>Make sure the following files reside in your system directory (for example, in Windows 2000, this directory is C:\WINNT\system32)</p> <ul style="list-style-type: none"> ● msxml4.dll ● msxml4a.dll ● msxml4r.dll <p>If they do not reside there, reinstall eWebEditPro+XML.</p>
The Web page that loads the editor does not load a schema.	<p>See "Loading Schemas" on page 666.</p> <p>Also, go to the XML schemas sample by clicking the Windows Start button and following this path: Programs > Ektron > ewebeditpro5 >Samples >XML > Schemas> XML Schemas. If the XML tags on this page load properly, the XML page you are using is not loading schemas.</p>

Working with Attributes

Displaying Attributes

You can display an element's attributes within the tag section. An attribute's schema definition determines the type of response that is allowed (see following table), as well as a default value, whether a response is required, and the maximum and minimum number of items that can be inserted. An attribute's definition in the XML information determines its formatting, such as font type, size, color and background color.

How attribute is defined in schema	How attribute displays
Generic text. For example <pre><xs:attribute name="note" type="xs:string"/></pre>	A text area field. For example 
A set of values. For example <pre><xs:attribute name="city" default="Fresno"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="Boston"/> <xs:enumeration value="Fresno"/> <xs:enumeration value="Dallas"/> <xs:enumeration value="Chicago"/> </xs:restriction> </xs:simpleType> </xs:attribute></pre>	A list box. The user selects one of the values. For example 
A Boolean value. For example <pre><xs:attribute name="visible" default="true"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="true"/> <xs:enumeration value="false"/> <xs:enumeration value="yes"/> <xs:enumeration value="no"/> <xs:enumeration value="1"/> <xs:enumeration value="0"/> </xs:restriction> </xs:simpleType> </xs:attribute></pre>	A check box. For example 

Setting Default Values for Attributes

The `docxml` element data includes a child element, `xsd`, for defining a schema. For example

```
<xsd status="active" src="samples/xml/schemas/bookinfo.xsd" ns="urn:bookinfo"/>
```

Within the schema, use the `default` attribute to specify an attribute's default value. To continue the above example, the `bookinfo.xsd` schema would have the following data:

```
<xs:attribute name="note" type="xs:string" default="Ghost writer involved"/>
```

The note attribute's default value is **Ghost writer involved**.

If you do not declare a default for *generic text*, nothing appears in the field that captures the attribute content.

If you do not declare a default for a *boolean* or a *set of valid values*, the first item on the list of values is the default.

Specifying a Number of Attribute Occurrences

Within the schema, you can use the `minOccurs` and `maxOccurs` properties to specify a minimum and maximum number of times that a user can insert an attribute within an element. The default value for `maxOccurs` is 1.

If the user is not required to insert an attribute, enter `minOccurs="0"` into the attribute's definition. For example

```
<xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="character" type="xs:string"/>
    <xs:element name="placename" type="xs:string"/>
</xs:choice>
```

In contrast, to let a user insert an unlimited number of attributes, enter `maxOccurs="unbounded"` into the attribute's definition. For example

```
<xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="character" type="xs:string"/>
    <xs:element name="placename" type="xs:string"/>
</xs:choice>
```

Requiring an Attribute Value

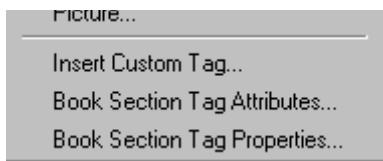
Within the schema, you can use the `use="required"` attribute to force the user to enter an attribute value. For example

```
<xs:attribute name="city" use="required">
```

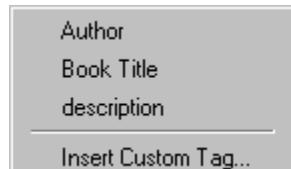
How Users Insert Attribute Tags

When a user inserts attribute tags for an element, he follows these steps.

1. Move the cursor to the element.
2. Right click the mouse. The context menu appears.



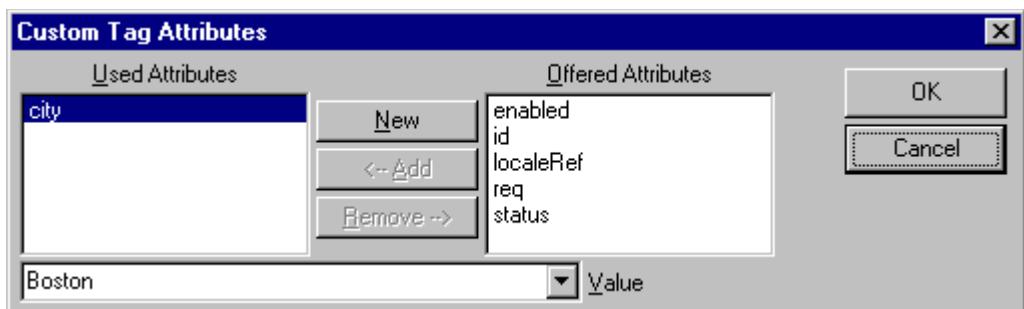
3. Select **Insert Custom Tag**. A list of child elements of the original element appears.



4. Select the child element to insert.
5. The Custom Tag attributes dialog appears, showing the attributes of the selected child element. Any required elements appear automatically in the Custom Tag attributes dialog and in the editor content.

NOTE

When displaying the list of attributes and their values, **eWebEditPro+XML** refers to the schema identified in the XML information.



The top left pane (**Used Attributes**) displays attributes that have already been added to the element. The top right pane (**Offered Attributes**) displays attributes of the element that have not been added.

You can remove a used attribute by selecting it and pressing **Remove**. (You cannot remove a required attribute.) You can add an unused attribute by selecting it and pressing **Add**. To change a used attribute's *value*, select it and use the drop down box in the **Value** field to make your choice.

You can even add an attribute not assigned to the element. To do this, press the **New** button.

6. Change any attribute values as desired.
7. Press **OK**.

Note that a schema can limit the number of attributes that you can insert.

The developer can modify the above scenario by changing the attribute values of the `tagattrdlg` element of the `<customtag>` feature in the configuration data. The following table lists that element's attributes and how they affect the user's ability to insert custom tags.

This attribute of the <code>tagattrdlg</code> element	Determines whether or not
<code>enabled</code>	the Custom Tag attributes dialog appears
<code>allownew</code>	the user can insert "new" tags (that is, tags that are not child elements or attributes of the current element). This attribute lets the user define tags not previously defined in the <code>xmlinfo</code> data or in the content.
<code>allowdelete</code>	the user can remove used attributes (users cannot remove a required attribute)
<code>allowchange</code>	the user can change an attribute's value

Validating XML Content

Validation Files

You can use the following files to validate content against a schema or DTD. The validation can occur in real time (schemas only) or externally to the editor (schemas and DTDs).

Schema Files

File	Description
xmlinfo.xsd	<p>Schema that defines the XML/Custom Tag XML configuration file settings. It provides a strict format for defining XML and custom tags.</p> <p>This file can validate XML/Custom Tag definitions contained in their own stream or file.</p> <p><i>See Also:</i> "The XML DTD and Schema" on page 678</p>
config.xsd	<p>Schema that defines the standard editor configuration settings. This contains custom tag element definitions for the XML and non-XML versions of the editor.</p> <p>Note that some validators might find errors when validating config.xml against config.xsd because some attributes have no value by default.</p>

XML File

File	Description
xmlinfo.xml	<p>Sample XML data illustrating the XML/Custom Tag XML configuration file settings.</p> <p>This sample uses a set of tags contained in a book schema.</p> <p><i>See Also:</i> "Assigning XML Data in a Script" on page 649</p>
config.xml	Defines the configuration data.

Validating with Schemas

You can validate editor content against a schema as a user inserts tags as well as externally. (To learn more about schemas, see "Working with Schemas" on page 666.)

Validating Upon Tag Insertion

You can prevent users from inserting invalid elements and attributes. By doing so, you are effectively performing real time validation, because users can only enter tags that are declared in the schema.

Preventing Entry of New Elements

To ensure that users can only insert valid elements, remove the ability to insert new tags. To do this, set the `allownew` attribute of the `taginsdlg` element to **false**.

```
<customtag>
  <taginsdlg allownew="false"/>
```

When you do, the context menu omits the Insert Custom tag option. As a result, users can only enter child elements that are declared in the schema for the current element.

Preventing Entry of New Attributes

To ensure that users can only insert valid attributes, remove the ability to insert new tags. To do this, set the `allownew` attribute of the `tagattrdlg` element to **false**.

```
<customtag>
  <tagattrdlg allownew="false"/>
```

When you do, the context menu omits the Insert Custom tag option. As a result, users can only enter attributes that are declared in the schema for the current element.

Validating Outside the Editor

You can validate editor content against a schema external to the editor.

You can define a schema for validation in the XML data or by using the validation function. The schema does not need to be loaded before validation. If it is not, the editor loads it before validation occurs.

If a schema is loaded before validation, the schema file value can be empty. In this case, the namespace value is used to select the validation schema.

If a schema is specified but the namespace value is empty, the editor tries to find the schema in the list of loaded schemas. If the schema is not loaded, it is loaded using either the path or the internal namespace as the URI.

If the schema and namespace are blank, no schema is selected for validation. In this case, the validation information must be defined in the XML data.

Below is an example of how to validate with a schema in JavaScript.

```
function ValidateXML(sEditorName, sXMLString, sSchemaPath, sNSTarget)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    objXmlDoc.Validate(sXMLString, sSchemaPath, sNSTarget);
    if(objXmlDoc.getPropertyInteger("ErrorCode") == 0)
    {
        alert("Passed!");
    }
    else
    {
        alert("ERROR: " + objXmlDoc.getPropertyInteger("ErrorCode") + " : " +
              objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

See Also: ["Method: Validate" on page 132](#)

Working with Schemas

NOTE

Instructions for how to create a schema are beyond the scope of this documentation. Many books and Web sites are devoted to this topic, including <http://www.w3schools.com/schema/default.asp>.

Schemas offer much flexibility and power to define XML data. Use schemas to define the structure of the XML elements and attributes used on a Web page. You can use a standard schema, such as NewsML, or create your own.

NOTE

Use a Web page's XML information to describe the appearance of its elements and attributes. See Also: "Where to Define the Appearance of XML Tags" on page 648.

Schemas let you validate tags as a user tries to insert them as well as externally.

Any number of schemas can be loaded using the loadsch element of the XML information. Only one schema is active at a time.

See Also: "Loadsch Element" on page 699

This section also describes the following topics:

- [Loading Schemas](#)
- [Selecting Schemas](#)
- [Validating with Schemas](#)
- [Extracting Element Information](#)

Loading Schemas

You can use the xmlinfo file to specify which schemas to load and which namespace to use. Here is an example of how these are defined.

```
<docxml enabled="true" reqfill="true" showroot="true">
    <!-- We seem to need the full path for the XSLT files, including the server. -->
    <transform onload="" onsave="/" />
    <schemas enabled="true">
        <!-- A ns value of "" means a namespace will be determined for the schema
            based on the path or an internal namespace.
        The status values are:
            active - used for offering valid options to user
            idle - stored for later selection or use (default)
            disabled - entry is ignored and not loaded
        - If more than one entry is selected as active then the first entry is used.
        - If no entry is active then the first schema is used.-->
        <xsd status="active" src="C:\Inetpub\wwwroot\somdev\DebugTags.xsd" ns="" />
        <xsd src="C:\Inetpub\wwwroot\somdev\books.xsd" ns="urn:books" />
    </schemas>
</docxml>
```

Here is an example of how to load a schema in JavaScript during the edit session.

```
function LoadSchema(sEditorName, sSchemaLocation, sNameSpace)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    objXmlDoc.AddSchema(sSchemaPath, sNSTarget);
    if(0 != objXmlDoc.getPropertyInteger("ErrorCode"))
    {
        alert("Error: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

See Also: ["Method: AddSchema" on page 54](#)

You can specify an `xmlInfo` stream to load schemas at initialization time so that the content's formatting will be clean to the user. Here is an example of how to do this on an HTML page.

```
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{
    // This loads the tag definitions, schemas, and transformations that we will use.
    eWebEditPro.parameters.xmlInfo = "schemainfo.xml";
    eWebEditPro.create("MyContent1", "100%", 400);
}
//-->
</script>
```

You can enumerate all loaded schemas using JavaScript. Below is an example.

See Also: ["Property: ActiveSchema" on page 136](#)

```

function ListLoadedSchemas(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var strVals = objXmlDoc.AvailableSchemas(" | ");
    var strActive = objXmlDoc.getPropertyString("ActiveSchema");

    // The method objXmlDoc.AnySchemasLoaded() could also have been
    // used here to determine if any shemas are loaded, but checking
    // the length of the returned string will do the same thing, and
    // we have gone to the trouble of retrieving it, so we will use
    // it to check for any valid schemas.
    if(strVals.length > 0)
    {
        var aryVals = strVals.split(" | ");
        var idx = 0;

        for(idx = 0; idx < aryVals.length; idx++)
        {
            if(aryVals[idx] == strActive)
            {
                alert("Loaded Schema: " + aryVals[idx] + " ** Active **");
            }
            else
            {
                alert("Loaded Schema: " + aryVals[idx]);
            }
        }
    }
    else
    {
        alert("There are no schemas loaded.");
    }
}

```

Selecting Schemas

When a schema is selected, it is considered active. You select a schema through the `xmlinfo` file or JavaScript.

To select a schema in an `xmlinfo` file, list the schema within the `schemas` element and set its `status` attribute to "active".

```
<xsd status="active" src="C:\Inetpub\wwwroot\somdev\DebugTags.xsd" ns="" />
```

To select a schema in JavaScript, use its assigned namespace. Here is an example.

See Also: ["Property: ActiveSchema" on page 136](#)

```

function SelectSchema(sEditorName, sNameSpace)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();

    objXmlDoc.setProperty("ActiveSchema", sNameSpace);
    if(objXmlDoc.getPropertyInteger("ErrorCode") == 0)
    {
        alert("Schema successfully selected.");
    }
    else
    {
        alert("Error selecting schema: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}

```

When a schema is selected through the interface, a cleaning operation is automatically performed. As a result, the content is displayed according to the active schema.

Validating with Schemas

See ["Validating with Schemas" on page 664.](#)

Extracting Element Information

The following examples show how to extract different pieces of schema information: an attribute, and attribute value, and an element

See Also: ["Method: ElementAttributes" on page 72.](#)

```

function ShowValidAttributes(sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var strVals = "";

    strVals = objTag.ElementAttributes(" | ");
    if(strVals.length > 0)
    {
        var aryAttrs = strVals.split(" | ");
        for(idAttr = 0; idAttr < aryAttrs.length; idAttr++)
        {
            alert("Attribute: " + aryAttrs[idAttr]);
        }
    }
    else
    {
        alert("There are no attributes for this element.");
    }
}

```

See Also: "Method: ElementAttributeValues" on page 73

```
function ShowValidAttributeValues (sEditorName, sAttribute)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idData = 0
    var strVals = "";
    var strDefVal = "";

    strVals = objElem.ElementAttributeValues(sAttribute, " | ");
    if(0 != strVals.length)
    {
        var aryData = strVals.split(" | ");
        strDefVal = objTag.AttributeValueDefault(sAttribute);
        for(idData = 0; idData < aryData.length; idData++)
        {
            if(strDefVal == aryData[idData])
            {
                alert("Attribute Value: " + aryData[idData] + " [ default ]");
            }
            else
            {
                alert("Attribute Value: " + aryData[idData]);
            }
        }
    }
    else
    {
        alert("There are no defined values for " + sAttribute);
    }
}
```

See Also: "Method: ElementChildren" on page 74

```
function ShowValidElements (sEditorName)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var objTag = objXmlDoc.ActiveTag();
    var idx = 0;
    var strVals = "";

    strVals = objTag.ElementChildren(" | ");
    if(0 < strVals.length)
    {
        var aryVals = strVals.split(" | ");
        for(idx = 0; idx < aryVals.length; idx++)
        {
            if(objElem.MinChildElementCount(aryVals[idx]) > 0)
            {
                alert("Element: " + aryVals[idx] + " is required.");
            }
            else
            {
                alert("Element: " + aryVals[idx]);
            }
        }
    }
    else
    {
        alert("There are no child elements within '" + objElem.TagName + "' .");
    }
}
```

Externally Editing XML Tags

This section explains how to allow the external editing of XML tags and their attributes. The editing is initiated by detecting a double click on an XML tag. A double click event can launch scripts that allow editing of a tag, its attributes, and its content.

- retrieving and setting attributes
- retrieving attribute values
- determining if a tag is selected

Retrieving the Current XML Tag

To retrieve the current tag (that is, the tag closest to where the user is editing or selecting), use the XML data object to access the Object Interface.

See Also: “[XML Object](#)” on page 18

Below is a sample function that retrieves the object for a specific editor.

```
function GetXMLProcessor(sEditorName)
{
    // Usage: var objXml = GetXMLProcessor(sEditorName);
    return(eWebEditPro.instances[sEditorName].editor.XMLProcessor());
}
```

Next, use the Object Interface in a JavaScript function to examine or edit a tag's attributes. Below is a sample function that retrieves the current tag using the Object Interface.

```
function GetCurrentXMLTag(sEditorName)
{
    // Usage: var objTag = GetCurrentXMLTag(sEditorName);
    return(eWebEditPro.instances[sEditorName].editor.XMLProcessor().ActiveTag());
}
```

Note There is no way to enumerate outward or inward from the current tag.

Changing an XML Tag

Once you retrieve a tag, one of the things you can do is change it. Changing the Data object (CXMLData) does not directly change the tag in the editor. (There are many technical and practical reasons for this.) The Data object holds the changes until you are ready to apply them.

The following sample JavaScript code shows how to change a tag.

```
function SetTagAttributes(sEditorName, sAttributes)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        // Change the tag here
    }
}
```

```

        objXmlTag.setProperty("Attributes", sAttributes);
        objXmlDoc.ApplyTag(objXmlTag);
    }
}

```

Retrieving and Setting Attributes

Use the `Attributes` property to retrieve a tag's attributes. See Also: “[Property: Attributes](#)” on page 142.

The following script illustrates how to do this.

```

function GetTagAttributes(sEditorName)
{
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        return(objXmlTag.getPropertyString("Attributes"));
    }
}

```

Retrieved attributes appear in the format as seen in the tag, for example
`attr1="a" attr2="b" attr3="c".`

You must use this format when setting an Attribute's property. The following example script illustrates how to set attributes. It assumes the format is correct.

```

function SetTagAttributes(sEditorName, sAttributes)
{
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        return(objXmlTag.setPropertyString("Attributes", sAttributes));
    }
}

```

Using a Tag Object to Retrieve a Tag

You do not need a script function to retrieve the current tag. Instead, you can pass a tag object between functions to speed up processing and ensure that functions work with the same tag.

Delimiting and Un-Delimiting Attributes

A function is available to delimit attribute/value pairs. Here is an example.

```

function GetTagAttributesDelim(sEditorName)
{
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        var strAttrs = objXmlTag.getPropertyString("Attributes");
        return(objXmlTag.DelimitAttributes(strAttrs));
    }
}

```

If you use the above function to delimit attribute/value pairs, the resulting format is: attr1| "a" |attr2| "b" |attr3| "c" | . (The delimiter character is the pipe (|)). This makes parsing easier.

You can change a delimited list back to the format required for the Attributes property. The following script function does this.

```
function SetTagAttributesDelim(sEditorName, sAttrDelim)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        var sAttrs = objXmlTag.FormatDelimitedAttributes(sAttrDelim);
        objXmlTag.setProperty("Attributes", sAttrs);
        objXmlDoc.ApplyTag(objXmlTag);
    }
}
```

Retrieving Attribute Values

If the above mechanism of retrieving a full set of attribute values is more than required, other methods are available to retrieve attribute values for an active tag. The script below shows how to retrieve a tag attribute.

```
function GetSpecificAttribute(sEditorName, sAttribute)
{
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    if(true == objXmlTag.IsValid())
    {
        return(objXmlTag.GetTagAttribute(sAttribute));
    }
}
```

Determining if a Tag is Selected

You can use the `isMyTag` function to check if a specified XML tag has a tag name and whether the tag is selected.

```
function isMyTag(strTagName, objInstance, objXmlTag)
{
    return (objXmlTag.IsValid() && strTagName == objXmlTag.TagName
&& objInstance.editor.IsTagApplied(strTagName));
```

Example Usage: Reading an Attribute

```
var objInstance = eWebEditPro.instances[sEditorName];
var objXmlTag = objInstance.editor.XMLProcessor().ActiveTag();
if (isMyTag("mytagname", objInstance, objXmlTag))
{
    var strAttrValue = objXmlTag.GetTagAttribute("myattr");
```

Example Usage: Writing an Attribute

```
var objInstance = eWebEditPro.instances[sEditorName];
```

```

var objXmlDoc = objInstance.editor.XMLProcessor();
var objXmlTag = objXmlDoc.ActiveTag();
if (isMyTag("mytagname", objInstance, objXmlTag))
{
    objXmlTag.SetTagAttribute("myattr", "attribute value");
    objXmlDoc.ApplyTag(objXmlTag);
}

```

Modifying a Tag's Contents

Use the Content property to access the contents of any specific tag.

NOTE This functionality applies to all XML tag types *except* non-empty tags with floating icons, completely hidden tags, and protected ASP and script tags.

Retrieving Tag Contents

To retrieve a tag's contents, follow these steps.

1. Get a reference to the tag.
2. Extract the Content property.

This example displays a tag's contents in a text area field.

```

function ShowTagContents(sEditorName)
{
    var objTag = GetCurrentXMLTag(sEditorName);
    document.frmEditor1.ViewHTML.value =objTag.getPropertyString("Content");
}

```

Modifying Tag Contents

To modify a tag's contents, use the Content property. Follow these steps.

1. Retrieve a reference to the XML Object Interface (see "["Retrieving the Current XML Tag" on page 672](#)).
2. Retrieve a reference to the tag (see "["Retrieving the Current XML Tag" on page 672](#)).
3. Modify the Content property. The tag is not applied to the document at this point.
4. Use the ApplyTag method to insert the changes into the content.

WARNING!

When you modify a tag's contents, you change *all* contents within the tag, including any tags.

Below is an example of modifying a tag's contents.

```

function UseTagContents(sEditorName)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    var objTag = GetCurrentXMLTag(sEditorName);
    objTag.setProperty("Content", document.frmEditor1.ViewHTML.value);
    objXmlDoc.ApplyTag(objTag);
}

```

Implementing a Double Click Notification

eWebEditPro+XML's standard command mechanism (that is, the eWebEditProExecCommand function) is used for notification of a request for external editing.

Double clicking a tag sends the dblclicktag command to the client script. A script can act on the command notification. The text parameter contains the tag name.

The following script example shows how to implement a double click notification.

```
function eWebEditProExecCommand(sEditorName, strCmdName, strTextData, lData)
{
    if("dblclicktag" == strCmdName)
    {
        ProcessDoubleClickTag(sEditorName, strTextData, lData);
    }
}
function ProcessDoubleClickTag(sEditorName, strTextData, lData)
{
    alert("Clicked Tag: " + strTextData);
}
```

Implementing Double Click Action

When processing a double click, you can access the XML document functionality. Below is an example of the ProcessDoubleClickTag function.

```
function ProcessDoubleClickTag(sEditorName, strTextData, lData)
{
    var objXmlTag = GetCurrentXMLTag(sEditorName);
    alert("Tag name: " + objXmlTag.TagName);
}
```

Defining External Tag Functionality

To have external tag property functionality appear in the user interface, define the cmdtagextprop command in the configuration file.

See Also: ["Editing the Configuration Data" on page 312](#)

Once the command is defined, the option appears on the context menu. Here is how to define the command in the XML configuration information.

```
<customtag>
  ...
    <command name="cmdtagextprop">
      <image key="one"/>
      <caption>Edit Tag Properties</caption>
      <tooltiptext>External Edit Tag Attributes</tooltiptext>
    </command>
  ...
</customtag>
```

Script Activation of Double Click Command

A script can initiate an external double click command by sending the cmdtagextprop command to the editor. Then, the script acts on the dblclicktag as normal.

To help process the dblclicktag command sent from a script command, use the long parameter in the cmdtagextprop command. Any value placed into the long parameter is sent with the dblclicktag command. The script could then use it. The text parameter is replaced by the tag description, no matter what.

Below is a sample script generating the command.

```
function GenerateTagDoubleClick(sEditorName)
{
    var objEditor = eWebEditPro.instances[sEditorName];
    objEditor.editor.ExecCommand("cmdtagextprop", "", 999);
}
```

Modifying the Context Menu

If the cmdtagextprop command is defined in the configuration file, the option to externally edit the tag appears on the context menu. Below is an example of how to modify the Custom Tag feature settings in the configuration file to do this.

```
<customtag>
    <taginsdlg enabled="false" />
    <tagpropdlg enabled="false" />
    <tagattrdlg enabled="false" />
    <command name="cmdtagextprop">
        <image key="one" />
        <caption>Edit Tag Properties</caption>
        <tooltiptext>External Edit Tag Attributes</tooltiptext>
    </command>
    .
    .
</customtag>
```

If the user interface should allow only external editing of tags, disable the other Custom Tag functionality (described in ["Custom XML Dialog Boxes" on page 714](#)) to remove it from the interface.

The context menu also provides Section 508 compliance.

The Custom XML Tag DTD and Schema

The XML DTD and Schema

The following XML data type definition (xmlinfo.dtd) defines the structure of custom XML tags that you can create. This structure is also available as a schema in the xmlinfo.xsd file. Both files are installed to the directory to which you install **eWebEditPro+XML**.

```

<!--
ewebeditpro 4 XML Information DTD
Date: Jul 17, 2002
Copyright 2000-2002, Ektron, Inc.
-->
<!--XML Information-->
<!ENTITY % boolean "true | false | yes | no | 1 | 0">
<!ELEMENT customtag (tagdefinitions, docxml)>
<!ATTLIST customtag
    enabled (%boolean;) #IMPLIED
    tagsenabled (%boolean;) #IMPLIED
>
<!ELEMENT tagdefinitions (tagdefault, tagspec*)>
<!ATTLIST tagdefinitions
    enabled (%boolean;) #IMPLIED
    name CDATA #REQUIRED
    src CDATA #IMPLIED
>
<!ELEMENT tagdefault (glyph?, simtaglist?)>
<!ATTLIST tagdefault
    dstyle CDATA #IMPLIED
    astyle CDATA #IMPLIED
    ashow (%boolean;) #IMPLIED
    edit (%boolean;) #IMPLIED
    enabled (%boolean;) #IMPLIED
    render (%boolean;) #IMPLIED
    style CDATA #IMPLIED
    type (empty | nonempty | vertical | horizontal) #REQUIRED
    visible (%boolean;) #IMPLIED
>
<!ELEMENT tagspec (caption?, glyph?, simtaglist?)>
<!ATTLIST tagspec
    dstyle CDATA #IMPLIED
    astyle CDATA #IMPLIED
    ashow (%boolean;) #IMPLIED
    edit (%boolean;) #IMPLIED
    enabled (%boolean;) #IMPLIED
    name CDATA #REQUIRED
    render (%boolean;) #IMPLIED
    style CDATA #IMPLIED
    type (empty | nonempty | vertical | horizontal) #REQUIRED
    visible (%boolean;) #IMPLIED
>
<!ELEMENT caption (#PCDATA)>
<!ATTLIST caption
    enabled (%boolean;) #IMPLIED
    localeRef CDATA #IMPLIED
    visible (%boolean;) #IMPLIED

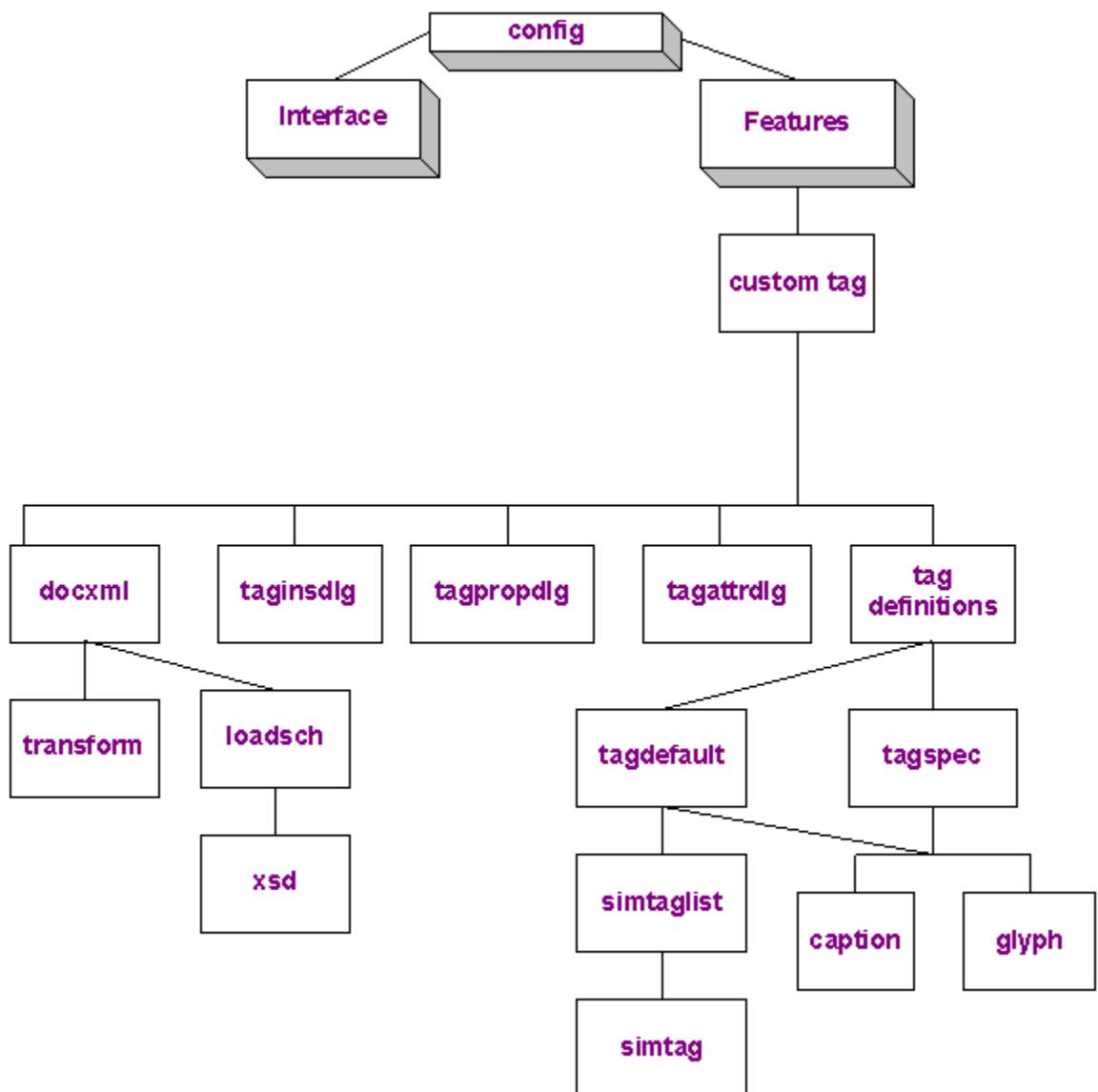
```

```

<!ELEMENT glyph EMPTY>
<!ATTLIST glyph
      height CDATA #IMPLIED
      src CDATA #REQUIRED
      visible (%boolean;) #IMPLIED
      width CDATA #IMPLIED
    >
<!ELEMENT simtaglist (simtag*)>
<!ATTLIST simtaglist
      enabled (%boolean;) #IMPLIED
      name CDATA #REQUIRED
    >
<!ELEMENT simtag (#PCDATA)>
<!ATTLIST simtag
      glyph CDATA #IMPLIED
      localeRef CDATA #IMPLIED
      name CDATA #REQUIRED
    >
<!ELEMENT docxml (transform?, loadsch?)>
<!ATTLIST docxml
      enabled (%boolean;) #IMPLIED
      reqfill (%boolean;) #IMPLIED
      showroot (%boolean;) #IMPLIED
    >
<!ELEMENT transform EMPTY>
<!ATTLIST transform
      onload CDATA #IMPLIED
      clean (%boolean;) #IMPLIED
      onsave CDATA #IMPLIED
    >
<!ELEMENT loadsch (xsd+)>
<!ATTLIST loadsch
      enabled (%boolean;) #IMPLIED
    >
<!ELEMENT xsd EMPTY>
<!ATTLIST xsd
      enabled (%boolean;) #IMPLIED
      status (idle | active | disabled) #IMPLIED
      src CDATA #REQUIRED
      ns CDATA #IMPLIED
    >
  
```

Custom Tag Elements

Below is a chart of the elements in this DTD and schema followed by an alphabetical summary of them.



Alphabetical Table of Custom Tag Elements

Element	Description	For more information, see
caption	Specifies the description of the tag presented to the user	"Caption Element" on page 692

Element	Description	For more information, see
customtag	The root element	"CustomTag Element" on page 683
docxml	Inserts required elements and attributes upon insertion of any element.	"Docxml Element" on page 696
glyph	Describes a glyph, or icon, associated with a tag	"Glyph Element" on page 693
loadsch	Contains a list of schemas to load	"Loadsch Element" on page 699
simtaglist and simtag	Allow custom tags to inherit attribute values from other tags	"Simtaglist Element" on page 694
tagattrdlg	Controls the Custom Tag attribute dialog.	"tagattrdlg" on page 702
tagdefault	Defines default values for all tags.	"Tagdefault Element" on page 693
tagdefinitions	Can identify external file that stores custom XML tag data	"Tagdefinitions Element" on page 683
taginsdlg	Controls the Insert Custom Tag dialog.	"taginsdlg" on page 701
tagpropdlg	Controls the Tag Properties dialog.	"tagpropdlg" on page 703
tagspec	Specifies a tag's appearance	"Tagspec Element" on page 684
transform	Identifies an XSLT file to use when loading or saving content. Also, whether the content should be cleaned when it is being loaded and/or saved.	"Transform Element" on page 698
xsd	Maintains a list of schemas to load	"XSD Element" on page 700

The rest of this section provides details about each element.

CustomTag Element

This element contains the definitions used by the Custom Tag feature.

Element Hierarchy

```
<config>
  <features>
    <customtag>
```

Child Elements

tagattrdlg, taginsdlg, tagpropdlg, command, cmd, tagdefinitions, docxml

Attributes

Attribute	Description
enabled	If false , the feature is disabled.
tagsenabled	If false , custom tag functionality is available to protect custom tags, but all custom tags are hidden and no interface is offered to the user.

Tagdefinitions Element

The tagdefinitions element affects the overall functionality of the custom tags feature.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <tagdefinitions>
```

Child Elements

tagdefault, tagspec

Attributes

Attribute	Description
enabled	Set to true if you want the XML data to be read. Otherwise, set to false .

Attribute	Description
name	The name of the tag; not used.

Tagspec Element

The tagspec element lets you specify that appearance of a custom tag.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <tagdefinitions>
        <tagspec>
```

Child Elements

[caption](#), [glyph](#)

Attributes

Attribute	Description
enabled	If true, data is kept between tags; if false, data is discarded
name	The name of the tag being defined
type	See "Types of XML Tags" on page 686
visible	If true , the tag appears; if false , the tag does not appear
edit	reserved - currently not implemented

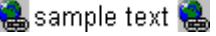
Attribute	Description
style	<p>The style definition for the <i>tag section</i> of an element (usually the same as a field label). See illustration below.</p> <div data-bbox="756 355 1281 456" style="background-color: #e0e0e0; padding: 5px; border: 1px solid black; margin-bottom: 10px;"> <p>This article applies to: tag section</p> <p>* eWebEditPro data section</p> </div> <p>You can use the CSS style attribute syntax to format text that prompts the user for input. For more information on this syntax, see http://www.w3.org/TR/html401/present/styles.html</p> <p>To have text appear in the tag section of an attribute, use the caption sub-element (see "Caption Element" on page 692.)</p>
dstyle	<p>The style definition for the <i>data section</i> of an element (usually the same as a field).</p> <p>Use the CSS style attribute syntax to format text that the user inputs. For more information on this syntax, see http://www.w3.org/TR/html401/present/styles.html</p> <p>Inheriting Style Directives</p> <p>If the dstyle attribute does not define all style directives defined in the style attribute, the value of any undefined directive is inherited from the style attribute.</p> <p>For example, if the style section is defined like this:</p> <pre>style='font-family:arial; font-weight:bold; background-color:#cccccc; border:solid blue 1pt; margin:2px; width:95%;'</pre> <p>and the dstyle section is defined like this:</p> <pre>dstyle='font-weight:normal; background- color:white; padding:4px'</pre> <p>the dstyle will use arial font, because font is not defined in the dstyle definition, and arial is defined for the style definition.</p>
astyle	<p>The style definition for the <i>tag section</i> of an attribute.</p> <p>Use the CSS style attribute syntax to format the attribute name. For more information on this syntax, see http://www.w3.org/TR/html401/present/styles.html</p> <p>See Also: "Inheriting Style Directives" on page 685</p>

Attribute	Description
ashow	A boolean attribute. If set to true , attributes are shown with the description. If false, attributes are not shown but can be accessed through the attributes dialog if it is enabled. The default value is true .
render	reserved - currently not implemented

Types of XML Tags

Custom XML tags are one of these four types.

Tag Type	Description	Example
Vertical	Typically used for large blocks of text, such as paragraphs. This is a blocking tag. See Also: "Vertical and Horizontal Tags" on page 688	<p>News Lines</p> <p>The world's first paying space tourist arrived at the International Space Station early Monday, within hours of space shuttle Endeavour's departure. California millionaire Dennis Tito and two cosmonauts hooked up to the orbiting outpost two days after they rocketed into space from the Russian launch facility in Kazakhstan. About 90 minutes after arriving, Tito and the two cosmonauts floated aboard the station and were welcomed by its crew. "I love space," Tito said with a wide smile.</p>
Horizontal	Typically used for smaller blocks of text, such as a field. This is a blocking tag. See Also: "Vertical and Horizontal Tags" on page 688	<p>Provider MSNBC STAFF</p>

Tag Type	Description	Example
Non-empty	<p>Surrounds small amount of editor content; typically indicated by background shading or an icon.</p> <p>This is a non-blocking tag.</p> <p>See Also: "Nonempty Tags" on page 688</p>	 
Empty	<p>Accepts no content.</p> <p>This is a non-blocking tag.</p> <p>See Also: "Empty Tags" on page 688</p>	<p>A tag that represents today's date</p>

These types can be divided into two groups: blocking and non-blocking.

Blocking Tags

A blocking tag always starts on a new line. It can be thought of as two dimensional; it has width and height. In HTML, blocking tags include P (paragraph) and TABLE.

Blocking tags are used to create a boundary or box in which to insert content. Blocking tags may contain other blocking and non-blocking tags. The vertical and horizontal types are blocking tags.

Non-blocking Tags

A non-blocking tag continues on the same line of text. It may encompass text that wraps across multiple lines, but there is a linear start and stop to the tag. Non-blocking tags include <A> (hyperlink), , and .

Non-blocking tags typically mark words and phrases with meta information. In the same manner that foreign words are italicized, words can be tagged with information about what they are rather than just their formatting.

For example, instead of using the HTML tag to mark a foreign word, you can use your own custom tag, say <foreign>. Then, you could write a query or XSL Transform (XSLT) to look up all foreign words in a dictionary and include their definitions in the document. This could be particularly helpful with legal, medical or technical documents.

Non-blocking tags can contain other non-blocking tags but cannot contain other blocking tags, including standard HTML blocking tags. For example, you cannot put a <P> tag within a non-blocking custom tag. The nonempty and empty types are non-blocking tags.

Now let's look at each type in more detail.

Vertical and Horizontal Tags

The vertical and horizontal types appear as a rectangular block. The block has two sections: a tag section where the tag name appears (optionally), and a data section where the author enters the content that belongs within that block.

This article applies to: **tag section**

* **eWebEditPro** **data section**

The tag name can appear either above (vertical) or to its left (horizontal). The vertical orientation is suitable for writing a few long and wide blocks of content. The horizontal orientation is best when there are several smaller content blocks. This format is similar to filling out a form.

You can mix vertical and horizontal types within a single document.

See Also: "Sample Vertical Tag" on page 689, "Sample Horizontal Tag" on page 690

Nonempty Tags

The nonempty type is non-blocking. That is, it does not cause the text to continue to the next line.

The nonempty tag has two basic looks: styled and glyphs (a small icon graphic). When a glyph is not used, the style is applied to the text within the nonempty tags. Typically the style affects the appearance of the text by making it italic, bold or by changing the background color.

When using glyphs, the style is ignored and the glyph appears at the beginning and ending tags. The glyphs can provide a pictorial clue about the tag, for example, an image of a person to denote an author or a calendar to denote a date.

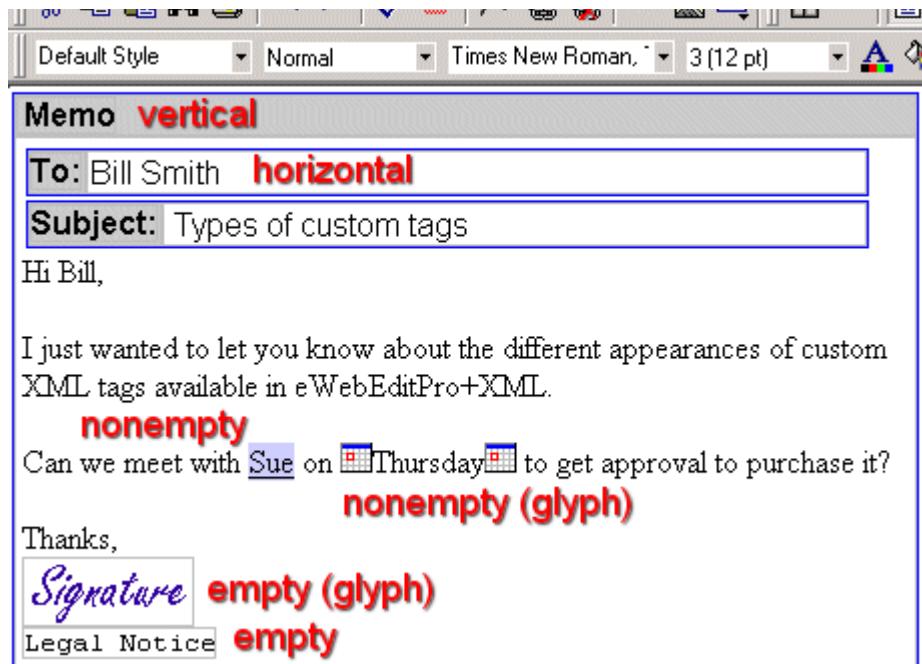
See Also: "Sample Nonempty Tag" on page 690, "Sample Nonempty Tag (glyph)" on page 690

Empty Tags

The empty type is unique in that it does not enclose any other text (that is why it is empty). An empty tag is presented by either its name or a glyph. The glyph can be any size and may be used as a place holder. For example, it may represent a UPC bar code that will be generated later. The HTML (image) and <HR> (horizontal rule) tags are examples of empty tags.

See Also: "Sample Empty Tag" on page 691

Finally, let's see some examples.



Sample Vertical Tag

Example: The large "Memo" box that contains all the other tags.

Tag Specification: mymemo

NOTE

In this example, the default is vertical, so the mymemo tag is placed as a simtag (similar tag) within the tagdefault element. However, the tagspec element could have been used. Only one tagdefault element is allowed.

```
<tagdefault type="vertical" visible="true"
    style="font-family:arial; font-weight:bold; background-color:#cccccc; border:solid blue
    1pt; margin:2px; width:95%;"
    dstyle="font-family:arial; font-weight:normal; background-color:white; padding:4px">
    <!-- The simtaglist is a quick list of tags that follow the default items above.
        The only offered deviation is the glyph that can be specified. If no glyph is given
        then the default glyph is used. -->
    <simtaglist name="deftaglist">
        <simtag name="mymemo">Memo</simtag>
        <simtag name="myps">P.S.</simtag>
    </simtaglist>
</tagdefault>
```

Note that

- The type attribute is `vertical`
- The `style` attribute defines the appearance of the tag name or caption
- The `dstyle` attribute defines the content area

Sample Horizontal Tag

Examples: The "To:" and "Subject:" fields.

Tag Specification: `myto, mysubject`

```
<tagspec name="mysubject" type="horizontal" visible="true"
  style="font-family:arial; font-weight:bold; background-color:#cccccc; border:solid blue
  1px; margin:2px; width:95%;">
  dstyle="font-family:arial; font-weight:normal; background-color:white; padding:2px;
  width:95%;">
  <!-- If a caption is not given, the name of the tag is used as the caption. -->
  <caption>Subject:</caption>
  <simtaglist name="deftaglist">
    <simtag name="myto">To:</simtag>
    <simtag name="mycc">Cc:</simtag>
  </simtaglist>
</tagspec>
```

Note that

- The type attribute is `vertical`
- The `style` attribute defines the appearance of the tag name or caption
- The `dstyle` attribute defines the content area

Sample Nonempty Tag

Example: The word "Sue". Your application could detect the `<myname>` tag around the word Sue, look the name up in a company directory, and automatically notify her about the meeting.

Tag Specification: `myname`

```
<tagspec name="myname" type="nonempty" dstyle="background-color:#cccff; text-
  decoration:underline;">
  <caption>Name</caption>
</tagspec>
```

Note that

- The type attribute is `nonempty` and no glyph is defined.
- The `dstyle` attribute defines the appearance of the text within the tag. In this case, a light blue background with underlining.

Sample Nonempty Tag (glyph)

Example: The word "Thursday". Your application could detect the `<mydate>` tag around the day of the week and, knowing the date the memo was written, automatically update the appointment calendars for Bill, Sue and the author.

Tag Specification: `mydate`

```
<tagspec name="mydate" type="nonempty" dstyle="background-color:transparent;">
  <caption>Date</caption>
  <glyph src="[eWebEditProPath]/tagcalendar.gif" visible="true" width="0" height="0" />
</tagspec>
```

Note that

- The type attribute is `nonempty` and a glyph is defined.
- The `dstyle` attribute is ignored.

Sample Empty Tag

Example: The "Legal Notice." Your application could replace the `<mylegal>` tag with your company's standard legal disclaimer. This spares the author from copying the legal notice.

Tag Specification: `mylegal`

```
<tagspec name="mylegal" type="empty" visible="true" style="font-family:Courier New;
  background-color:white; font-size:10pt;">
  <caption>Legal Notice</caption>
</tagspec>
```

Note that

- The type attribute is `empty` and no glyph is defined
- The visible attribute is `true` to show the tag name
- The style attribute controls how the tag name (or caption in this case) appears

Sample Empty Tag (glyph)

Example: The "Signature." Your application could replace the `<mysignature>` tag with a graphic of the author's signature and also include their digital signature.

Tag Specification: `mysignature`

```
<tagspec name="mysignature" type="empty" visible="false" style="background-color:transparent">
  <caption>Signature</caption>
  <glyph src="[eWebEditProPath]/tagsignature.gif" visible="true" width="0" height="0" />
</tagspec>
```

Note that

- The type attribute is `empty` and a glyph is defined.
- The visible attribute is `false` to hide the tag name--only the glyph shows.
- The style attribute is ignored.

Below is the XHTML with custom tags for the example. Because it is XHTML, it is ready to be parsed and processed by an XML parser or transform given an XSLT.

```
<mymemo>
<myto>
<p><font face="Arial, Helvetica">Bill Smith</font></p>
</myto>
<mysubject>&#160;Types of custom tags</mysubject>
```

```

<p>Hi Bill,</p>
<p>&#160;</p>
<p>I just wanted to let you know about the different appearances of custom XML
tags available in eWebEditPro+XML.</p>
<p>&#160;</p>
<p>Can we meet with <myname>Sue</myname> on <mydate>Thursday</mydate>&#160;to
get approval to purchase it?</p>
<p>&#160;</p>
<p>Thanks,</p>
<p><mysignature /></p>
<p><mylegal /></p>
</mymemo>

```

Caption Element

This element provides a description of the element presented to the user. The value of the caption appears in the tag section, which is on top for a vertical tag and on the left for a horizontal tag.

This article applies to: **caption**
* **eWebEditPro**

Element Hierarchy

```

<config>
  <features>
    <customtag>
      <tagdefinitions>
        <tagdefault>
          <caption>

        <tagspec>
          <caption>

```

Attributes

Name	Type	Description
enabled	Boolean	If true, the information contained in this element is loaded.
localeRef	String	If this version of eWebEditPro+XML will be in several languages, enter the locale code to describe the caption. See Also: "Locale Files" on page 266

Glyph Element

This element describes a glyph, or icon, that can be used to represent the tag to the user (see illustration below).



Element Hierarchy

```
<config>
  <features>
    <customtag>
      <tagdefinitions>
        <tagdefault>
          <glyph>

        <tagspec>
          <glyph>
```

Attributes

Name	Type	Description
height	integer	The height to use when displaying the glyph. If this attribute is omitted or has a value of 0, the actual value of the image is used.
src	string	The location of an image file (typically, a .gif) to display as the glyph. This should be a location over the internet or intranet.
visible	boolean	If false , the glyph does not appear. The same result occurs when there is no path to a file.
width	integer	The width to use when displaying the glyph. If this attribute is omitted or has a value of 0, the image's actual value is used.

Tagdefault Element

This element defines default attribute values that determine a tag's appearance if a tag is not defined. For example, if you insert a custom tag named `<book>` but do not define it in the configuration data, its attribute values are obtained from the `tagdefault` element.

System Default Values

If only some of a tag's attribute values are defined in a custom tagspec definition, eWebEditPro+XML applies a system default definition to the undefined attributes.

If the system defaults were implemented in XML, they would look like this.

```
<tagdefault enabled="true" type="vertical" visible="true" render="true" edit="true"
    style="font-family:arial; font-weight:bold; background-color:#cccccc; border:solid
        blue 1pt; margin:2px; width:95%;"
    dstyle="font-family:arial; font-weight:normal; background-color:white; padding:4px">
<caption enabled="true" localref="" visible="true"></caption>
<glyph src="" visible="false" width="0" height="0"/>
</tagdefault>
```

Although the visible attribute is defined in the DTD for the `caption` element, it is ignored and always set to **true** in this context.

Element Hierarchy

```
<config>
    <features>
        <customtag>
            <tagdefinitions>
                <tagdefault>
```

Child Elements

`simtaglist`

Attributes

`tagdefault`'s attributes are the same as the attributes you can define for the `tagspec` element. See "["Tagspec Element" on page 684](#)".

Simtaglist Element

The `simtaglist` and `simtag` elements reduce the effort required to create and maintain custom XML data for tags that share similar attributes. `simtaglist` and `simtag` also result in a smaller XML custom tags file, which reduces the time required to load the editor.

How the `simtaglist` and `simtag` Elements Work

Instead of defining each attribute for each tag, you can define tags and then use the `simtag` element to apply the attributes of those tags to similar tags. You can use `simtag` as long as the tags in the `simtag` list only differ in their definition of the `glyph` and `localeRef` attributes. If other attributes of the tags differ from the original tags, you cannot use `simtag` -- you must fully define the second set of tags.

Element Hierarchy

```
<config>
    <features>
```

```

<customtag>
  <tagdefinitions>
    <tagdefault>
      <simtaglist>

      <config>
        <features>
          <customtag>
            <tagdefinitions>
              <tagspec>
                <simtaglist>

```

Child Elements

Name	Required
Simtag (see "Simtag Element" on page 695)	any amount

Attributes

Attribute	Type	Description
enabled	Boolean	If false, the data contained in this element is not read.
name	string	The name of the list of tags.

Simtag Element

See Also: "Simtaglist Element" on page 694

Element Hierarchy

```

<config>
  <features>
    <customtag>
      <tagdefinitions>
        <tagdefault>
          <simtaglist>
            <simtag>

<config>
  <features>
    <customtag>

```

```
<tagdefinitions>
  <tagspec>
    <simtaglist>
      <simtag>
```

Child Elements

Sub-element	Description
#Text	The tag's caption if the locale element is not found in the localization file.

Attributes

Attribute	Type	Description
glyph	string	The location of the glyph to use, if the glyph defined in the parent element will not be used.
localeRef	string	The translation key to use to retrieve the description of the tag from the appropriate localization file. See "Modifying the Language of eWebEditPro+XML" on page 265.
name	string	The name of the custom tag. Required.

Docxml Element

This element lets you configure **eWebEditPro+XML** to insert all required elements and attributes of any element upon its insertion.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <docxml>
```

Child Elements

transform, loadsch

Attributes

Attribute	Type	Description
enabled	Boolean	Whether the feature is enabled. The default is true .
reqfill	Boolean	If this value is true (or not specified) and the user inserts an element contained within a loaded schema, all required elements within the inserted element are also inserted. If the automatically inserted elements have required elements, they are also inserted. If the elements have required attributes, these are included with the elements with either their default values, the first value in their value list, or as an empty value. If this is false , only the selected element is inserted.

Attribute	Type	Description
showroot	Boolean	<p>Specifies whether the root element of a pure XML document is displayed in WYSIWYG mode. Also, specifies whether the root tag is retrieved with the body.</p> <p>The site administrator may not want the user to see the root element to ensure that the user does not place values outside it, thus enhancing the possibility that the document validates.</p> <p>Users sometimes add items at the end of a document. Because the end of a document is outside the closing root tag, tags inserted outside the root element are illegal.</p> <p>If the root tag does not display, a user cannot add an item outside it in WYSIWYG mode. In this case, a user can only add an item outside the root tag when in Source View with the entire document set to display (that is, <code><viewas mode=whole></code>).</p> <p>The default setting for this attribute is true.</p> <p>Set to true if you want to display the root tag in WYSIWYG mode and in the source view of the body. If set to true, the root tag is retrieved with the body.</p> <p>Set to false to suppress the root tag in WYSIWYG mode and in the source view of the body. If set to false, the root tag is not retrieved with the body.</p> <p>Example: <code><docxml enabled="true" reqfill="true" showroot="true"></code></p> <p>Showing the Root Element in Source View</p> <p>If the root element is shown in source view (<code>showroot="true"</code>)</p> <ul style="list-style-type: none"> • the first element in the source view becomes the root element • if the header is retrieved, it does not contain the root element <p>Not Showing the Root Element in Source View</p> <p>If the root element is <i>not</i> shown in source view (<code>showroot="false"</code>)</p> <ul style="list-style-type: none"> • the root element cannot be set through the source view • the entire document, including the leading <code>?xml</code>, must be entered into the source view to modify the root value • use header retrieval methods to retrieve, modify, and set the root element <p>Of course, if an entire document is entered into source view, the entire document (including header and root elements) is changed.</p>

Transform Element

This element specifies the transformation files to use when loading or saving the content. This is an automatic operation, which means that the editor will always

show a transformed document and the client will always receive a transformed document.

This automated transformation only happens when a full document is loaded or retrieved. Partial documents or content that is pasted does not go through the transformation process.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <docxml>
        <transform>
```

Attributes

Attribute	Type	Description
enabled	Boolean	Whether the feature is enabled. The default is true .
onload	string	<p>The XSLT file to run when loading content. This transformation is performed on a document that is loaded into the editor through the <code>setDocument</code> method.</p> <p>This attribute does not work with the <code>setbodyhtml</code> and <code>getbodyhtml</code> methods.</p> <hr/> <p style="color: #0000ff;">Ektron's Knowledge Base contains several articles that describe using XSLTs. See http://www.ektron.com/support/ewebeditprokb.cfm?searchtext=xslt:</p> <hr/>
onsave	string	<p>The XSLT file to run when publishing (saving) content. This is a transformation on the contents when they are exported out of the editor.</p> <p>This attribute does not work with the <code>setBodyHtml</code> and <code>getBodyHtml</code> methods.</p>

Loadsch Element

This element contains a list of schemas to load.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <docxml>
        <loadsch>
```

Child Elements

xsd

Attributes

Attribute	Type	Description
enabled	Boolean	If true, the schema named in this element is loaded.

XSD Element

This element maintains a list of schemas to load. Any number of schemas can be loaded into the editor. Only one becomes active at a time.

Loaded schemas can be selected at run time. It is possible to select a specific schema in this list by using the element's status attribute.

Element Hierarchy

```
<config>
  <features>
    <customtag>
      <docxml>
        <loadsch>
          <xsd>
```

Attributes

Attribute	Type	Description
enabled	Boolean	If true, the information in this element is loaded.
status	string	The load status of the schema. There are three values: Idle - load and hang onto it Active - load and use it as the active schema. Disabled - do not load the schema into memory. It will not be available for selection.
src	string	The location of the schema file. This can be a relative or full path. This must be a file that is accessible to the editor. It cannot be a stream like XML. This attribute is required.

Attribute	Type	Description
ns	String	The namespace to assign to this schema. If this is blank, the schema location is used as the namespace. The location is the full path including the server. If the schema file specifies a namespace, that name is used instead of the name given here or the file location.

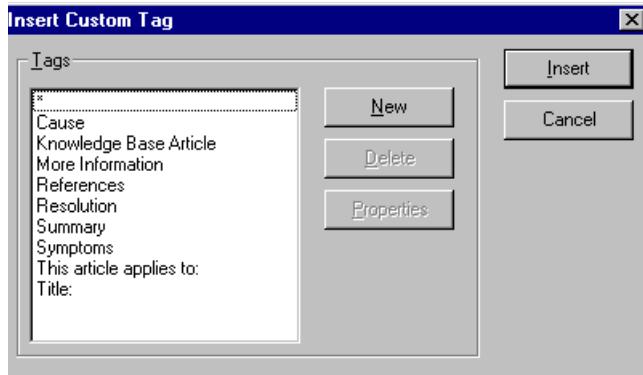
Elements that Control Dialog Boxes

This section describes three dialog boxes that let a developer or user enter custom tags, and edit the properties and attributes of custom tags.

By default, commands that launch these dialog boxes appear on the context menu. (To remove them, see "[Removing Commands from a Context Menu](#)" on page 256.) If no tag is selected, only the **Insert Custom** tag option is available on the context menu. If a tag is selected, the **Tag Attributes** and **Tag Properties** options are also available.

taginsdlg

This element lets you control the operation of the Insert Custom Tag dialog box, illustrated below.



Element Hierarchy

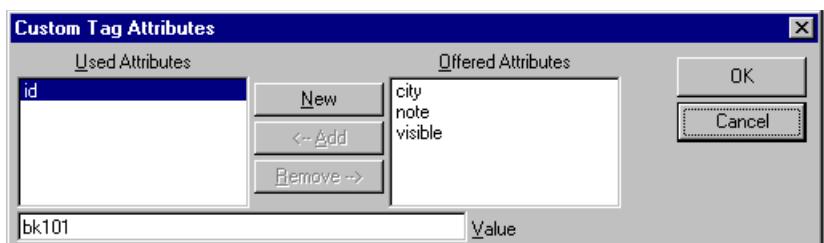
```
<config>
  <features>
    <customtag>
      <taginsdlg>
```

Attributes

Attribute	Type	Description
enabled	Boolean	If set to false , the data contained in this element is not read, and the dialog box is not available to the user.
allowprop	Boolean	If set to false , the user cannot edit the properties of a selected tag, and the dialog box's Properties button does not appear.
allowdelete	Boolean	If set to false , the user cannot delete an attribute, and the dialog box's Delete button does not appear.
allownew	Boolean	If set to false , the user cannot create a new tag, and the dialog box's New button does not appear.

tagattrdlg

This element lets you control the Custom Tag attribute dialog, illustrated below.

**Element Hierarchy**

```

<config>
  <features>
    <customtag>
      <tagattrdlg>

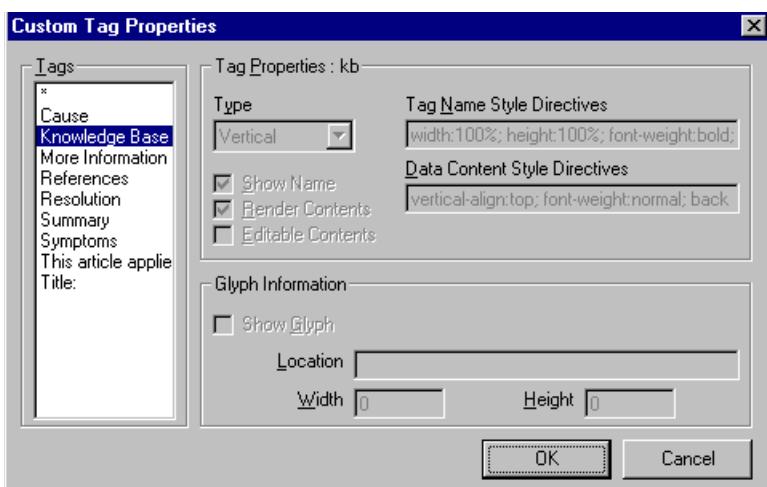
```

Attributes

Attribute	Type	Description
enabled	Boolean	If set to false , the data contained in this element is not read and the attribute dialog is not available to the user.
allowchange	Boolean	If set to false , the user cannot change any attribute values.
allowdelete	Boolean	If set to false , the user cannot delete an attribute, and the dialog box's Delete button is hidden.
allownew	Boolean	If set to false , the user cannot create a new attribute, and the dialog box's New button is hidden.

tagpropdlg

This element lets you control the Tag Properties dialog, illustrated below.

**Element Hierarchy**

```
<config>
  <features>
    <customtag>
      <tagpropdlg>
```

Attributes

Attribute	Type	Description
enabled	Boolean	If set to false , the data contained in this element is not read and the dialog box is not available to the user.
allowchange	Boolean	If set to false , the user cannot change any attribute values.
listall	Boolean	If set to false , only the current tag appears. Otherwise, all tags appear so that the properties can be seen and/or modified.

XML Commands, Methods and Parameters

Custom Tag Commands

The following commands allow the user and any client scripting to control custom tags and the custom tag dialog boxes.

`cmdcushtagprop`

`cmdcushidetags`

`cmdcushtaginsert`

`cmdcustshow`

`cmdcushtagattrs`

`cmdcushide`

`cmdcustapplytag`

cmdcushtagprop

Command: `cmdcushtagprop`

Description: Launches the tag properties dialog. This can be used as a toolbar button.

See Also: "The Custom Tag Properties Dialog" on page 716

Text: not used

Data: not used

cmdcushtaginsert

Command: `cmdcushtaginsert`

Description: Launches the Insert Custom Tag dialog. If content is selected when the command is executed, the tag is applied to that content. This can be used as a toolbar button.

See Also: "The Insert Custom Tags Dialog" on page 714

You can also write a script that causes this command to

- insert a specific element
- determine if default values and elements are automatically inserted with the element
- determine whether to display the attributes dialog to the user

Text: The name of the element to insert. If the name contains attributes, they are inserted with the tag.

Here are some sample Text Parameter values:

```
"book"
"book id='abc' "
"<book id='abc'>"
```

WARNING!

Any content declared with elements or attributes is ignored.

Data: Enter one of the following values to indicate how the command should

- insert required elements and attributes
- display the attributes dialog to the user (See Also: "[The Custom Tag Attributes Dialog](#) on page 715)

Value	Does command insert required elements and attributes?	Does command display the attributes dialog to user?
0	no	no
1	yes	no
2	no	yes
3	yes	yes

Example:

```
<script language="JavaScript1.2">
//!--
function PasteElement(sEditorName, sElement, iFillIn)
{
    var objElement = GetCurrentXMLTag(sEditorName);
    if(true == objElement.CanInsert())
    {
        eWebEditPro.instances[sEditorName].editor.ExecCommand("cmdcusttaginsert", sElement,
iFillIn);
    }
    else
    {
        alert("Can't insert at this location.");
    }
}
//-->
</script>

<input type="button" value="Paste Book Tag with Content" onClick="PasteElement('MyContent1',
'book', 1)">

<input type="button" value="Paste Book Tag without Content"
onClick="PasteElement('MyContent1', 'book', 0)">
```

cmdcusttagattrs

Command: cmdcusttagattrs

Description: Launches the Tag Attributes dialog. This can be used as a toolbar button.

See Also: ["The Custom Tag Attributes Dialog" on page 715](#)

Text: not used

Data: not used

cmdcustapplytag

Command: cmdcustapplytag

Description: Applies given tag and attribute information to the current selection. This is not meant as a toolbar button, but as a function for client scripting.

The text parameter to the command provides the tag information, name, and attributes in the normal tag format. The long parameter is not used.

For example:

```
objEditor.ExecCommand("cmdcustapplytag", "mya attr1='1' attr2='2'", 0);
```

Text: The custom tag information, including, the name and attributes of the tag.

This can be in XHTML format, or just a string with assignments. Here are two examples:

```
<mya attr1="1" attr2="2">
mya attr1="1" attr2="2"
```

Data: not used

cmdcusthidetags

Command: cmdcusthidetags

Description: Hides or shows all visible custom tags in the current content. This can be used as a toggle button on the toolbar.

The long parameter specifies whether the visible tags are hidden from the user.
For example:

```
objEditor.ExecCommand("cmdcusthidetags", "", 1);
```

Text: not used

Data: A value of 0 shows all tags defined with visible attributes, while a value of 1 hides visible tags.

cmdcustshow

Command: cmdcustshow

Description: Shows all defined, visible custom tags. This can be used for a toolbar button, although it is meant for scripting.

This command acts the same as the cmdcusthidetags command with the long parameter set to a value of 1.

Text: not used

Data: not used

cmdcusthide

Command: cmdcusthide

Description: Hides all defined visible tags from the user. This can be used for a toolbar button, although it is meant for scripting.

This command acts the same as the cmdcusthidetags command with the long parameter set to a value of 0.

Text: not used

Data: not used

Custom Tag Methods

eWebEditPro+XML exposes two methods that provide information about custom tags.

- “Method: TagCount” on page 127
- “Method: IsTagApplied” on page 99

Parameter Properties

xmlInfo

This parameter specifies the xmlinfo file to load when initializing the editor. The definitions, settings, and files specified in this data are loaded before the content is loaded. This provides a clean display of the content.

The item specified can be either a file or a stream. The xml can be assembled in the same fashion as the configuration data. If a file is given, it can be a relative path or a fully qualified path.

Here is an example of how this parameter is used.

```
<script language="JavaScript1.2">
<!--
if (typeof eWebEditPro == "object")
{
    eWebEditPro.parameters.xmlInfo = "xmlinfo.xml";
    eWebEditPro.create("MyContent1", "100%", 400);
}
//-->
</script>
```

See Also: “The `xmlInfo` Property” on page 649

XML Additions to Editor API

The editor API, which includes methods and attributes, is extended to support the functionality and objects offered for processing XML and schemas.

Property: XmlInfo

See “[Property: xmlInfo](#)” on page 164

Method: XMLProcessor() As XML Object

Description: This retrieves the interface into the object that contains all of the XML functionality. From here, the information can be set and retrieved from the XML document. Schemas and transformation files can be loaded and selected through the interface of the object returned with this method. This object is also used to retrieve a reference to the object that manages the current element.

If this method is called in the non-XML editor, a useless object is returned.

See Also: “[XML Object](#)” on page 18

Data Type: XML Object

Example: This example shows how the XMLProcessor method is used to load a schema.

```
function GetXMLProcessor(sEditorName)
{
    return(eWebEditPro.instances[sEditorName].editor.XMLProcessor());
}
function LoadSchema(sEditorName, sSchemaLocation, sNameSpace)
{
    var objXmlDoc = GetXMLProcessor(sEditorName);
    objXmlDoc.AddSchema(sSchemaPath, sNSTarget);
    if(0 != objXmlDoc.getPropertyInteger("ErrorCode"))
    {
        alert("Error: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

Modified API

Some existing API methods and attributes are modified to support the XML and schema documents. The actual API has not changed, but the functionality is improved to support these documents.

Method getDocument()

Normally, the getDocument method returns a full HTML or XHTML source code. If a full XML document is loaded, the getDocument method returns a full XML document. All header information is preserved and included with the document.

If the header is modified through the setHeadHTML method, the modified header is returned with the content to make a full XML document.

See Also: “[Method: getDocument](#)” on page 85

Method getHeadHTML

The getHeadHTML method was designed to retrieve the header to the HTML. If a full XML document is loaded, the editor goes into “XML Mode” where all operations assume that we are working with a full XML document.

The getHeadHTML method can be used to retrieve the header. It can be modified by the client scripting, if needed, and then placed back into the editor using the setHeadHTML method.

When the getDocument method is called, the full XML document is returned with the modified header information.

IMPORTANT NOTE

The powerful `showroot` attribute of the tagdefinitions element affects what is returned by this method.

- If the root element is shown, when the header is retrieved, it does not contain the root element.
- If the root element is not shown, the root element can be retrieved, modified, and set through the header retrieval methods.

See Also: "[Method: setHeadHTML](#)" on page 123; "["showroot"](#) on page 698

Method `setDocument`

Normally, this method is used to load an HTML document. With the XML product, it can be used to load a full XML document. In this case, this method supports the loading of pure XML data into the editor.

When a pure XML document is loaded into the editor, it goes into "Pure XML Mode" where it assumes that all operations are on a pure XML document.

All header information is maintained in the editor and return to the client through the getDocument method.

See Also: "[Method: setDocument](#)" on page 120

Method `setHeadHTML(strNewHead As String)`

Normally, this method modifies the header information for the HTML source. If a pure XML file is loaded, this method is used to modify the header information.

IMPORTANT NOTE

The setting of the `showroot` attribute is not as important for this method but still has some effect.

- If the root element is given, it is used if the root is not shown to the user.
- If the root element is not given when this method is used, the existing root information is preserved and used when the header and full content are retrieved.

See Also: "[Method: setHeadHTML](#)" on page 123; "["showroot"](#) on page 698

Implementing a Command that Inserts a Comment

This section provides an example of how to add a custom XML tag. It uses a `mycomment` button to execute the `jscomment` command, which inserts a custom comment tag into content.

When the comment button is pressed or the tag is double clicked, the showCommentDialog function displays the commentpopup.htm Web page dialog. The dialog prompts the user to enter a comment. When the user closes the dialog, the comment is inserted into the editor's content using a custom XML tag, as shown here.

```
<mycomment comment="This is the comment"/>
```

See Also: "Determining What Happens When the Button is Pressed" on page 711

The mycomment Button

The mycomment button is defined as shown below.

```
<button command="jscomment" />
```

By default, the button is not on a toolbar. To display it, modify the config.xml and either set allowCustomize="true" and customize the toolbar through the editor's user interface, or move the <button> element within one of the <menu> elements in config.xml.

See Also: "Defining the Toolbar" on page 230

When a user presses the button, the editor raises the onexeccommand event with the jscomment command, as defined in the default config.xml shown below.

```
<features>
  <external>
    <cmd name="jscomment" key="comment" ref="cmdComment" />
    :
  </external>
</features>
```

Determining What Happens When the Button is Pressed

The JavaScript below is executed when the button is pressed. Note that the default dblclicktag event handler is implemented to process double clicks of the mycomment tag within the content.

```
eWebEditProExecCommandHandlers["jscomment"] = function(sEditorName, strCmdName, strTextData, lData)
{
  showCommentDialog(sEditorName);
}

eWebEditProExecCommandHandlers["dblclicktag"] = function(sEditorName, strCmdName, strTextData, lData)
{
  var objXmlTag = eWebEditPro.instances[sEditorName].editor.XMLProcessor().ActiveTag();
  if ("mycomment" == objXmlTag.TagName)
  {
    showCommentDialog(sEditorName);
  }
}
function showCommentDialog(sEditorName)
{
  eWebEditPro.openDialog(sEditorName, "commentpopup.htm", "", "", "",
    "width=650,height=350,resizable,scrollbars,status,titlebar");
}
```

Modifying the Event Handler

This event handler function is defined in ewebeditproevents.js, but you should *not* modify ewebeditproevents.js. To modify an existing handler function or write your own, add it to customevents.js. Any functions defined in customevents.js override functions in ewebeditproevents.js.

See Also: "Implementing a Double Click Notification" on page 676

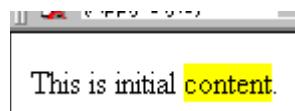
Using the Comment Sample

When you install **eWebEditPro+XML**, Ektron provides a sample Web page that illustrates how you can customize the comment feature. To launch the comment sample page, paste the following into your browser address window:

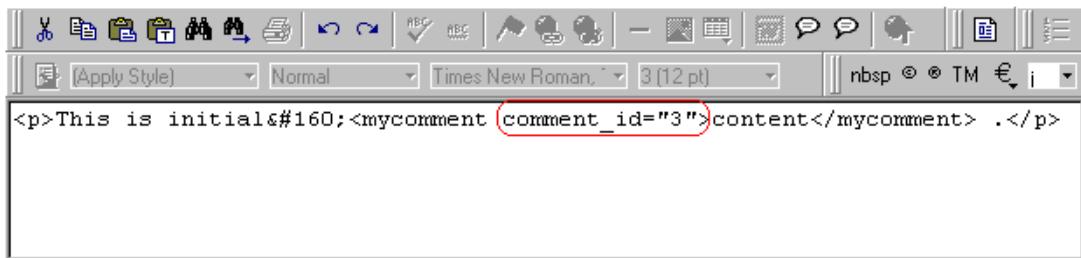
`http://localhost/ewebeditpro5/samples/asp/comment/index.asp`

The comment sample page has the following modifications:

- the icon that normally indicates that a comment has been applied does not appear
- the text to which the comment is applied has a yellow background (see illustration)



- instead of saving the comment with the Web page content, it is saved to a database (see illustration).



Some organizations may prefer saving a comment to a database (instead of in the content) because comments stored with the content can be viewed by anyone who presses the View as HTML button.

Modifications to Configuration Data that Enable Comment Sample

In order for the comment sample to work, you must modify the configuration data as follows

1. Add `<button command="jscomment1" />` to a standard menu.
2. Add `<cmd name="jscomment1" key="comment" ref="cmdComment" />` to the `<external>` section of the configuration data.
The `jscomment1` function is
3. Modify the `mycomment` custom tag element of the configuration data as follows (changed values are in red italics):

```
<tagspec name="mycomment" type="nonempty" visible="false" style="background-color:yellow" ashow="false">
```

```
    <caption>Comment</caption>
```

```
        <glyph src="[eWebEditProPath]/comment.gif" visible="false" width="16" height="16" />
    </tagspec>
```

NOTE For more information on the possible values for the `type` attribute, see "Types of XML Tags" on page 686.

Files Used by the Comment Sample

The comment sample uses three files to customize the comment feature.

- **commentfunc.asp** - saves the comment to a database
- **commentpopup.asp** - generates the custom popup window
- **index.asp** - displays the editor, and reacts to the `jscomment1` command by displaying the custom popup window

Custom XML Dialog Boxes

eWebEditPro+XML has three dialog boxes that can be used by a developer or a user to insert and modify custom tags, their attributes and properties. They are

- Insert Custom Tags
- Custom Tag Attributes
- Custom Tag Properties

"Elements that Control Dialog Boxes" on page 701 explains how to control the availability and functionality of the dialogs by editing their configuration data.

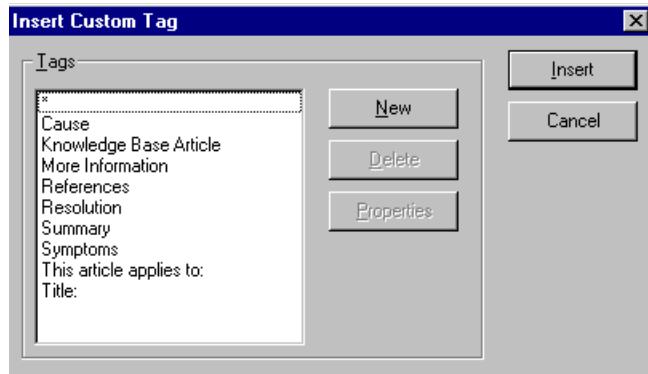
This section explains the user interface (that is, the fields and button) of these dialog boxes.

See Also: "Use Tag Properties Dialog as Diagnostic Tool" on page 725

The Insert Custom Tags Dialog

The Insert Custom Tags dialog (illustrated below) lets a user or developer insert a custom tag into the editor content. They can also add new tags, modify or delete existing tags, their attributes and properties.

This dialog is available on the context menu if the cmdcusttaginsert command exists in the configuration data.



The User Interface of the Insert Custom Tags Dialog

NOTE

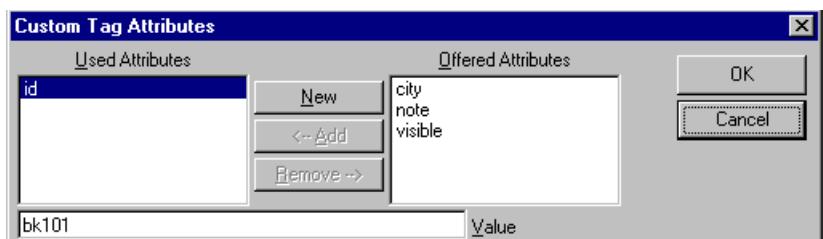
The attribute values of the taginsdlg element may suppress some of the buttons listed below. For more information, see "taginsdlg" on page 701.

Field or button	Description
Tags	<p>Select a tag from the list and press Insert to copy the custom tags into the editor content.</p> <p>When you press Insert, the Custom Tag Attributes dialog appears so that you can enter attributes for the tag.</p> <p>See Also: "The Custom Tag Attributes Dialog" on page 715</p>
New	<p>Create a new tag. If you do, a new line appears at the top of the Tags column for your entry.</p> <p>You can then click on the Properties button to launch the Custom Tag Properties dialog. Use this dialog to modify the new tag's properties before inserting it.</p> <p>Next, the Custom Tag Attributes dialog appears so that you can enter attributes for the tag.</p> <p>See Also: "The Custom Tag Attributes Dialog" on page 715</p>
Delete	Lets you delete the selected tag.
Properties	<p>Launches the Custom Tag Properties dialog. Use this dialog to modify the new tag's properties before inserting it.</p> <p>See Also: "The Custom Tag Properties Dialog" on page 716</p>
Insert	Copies the custom tags and its properties and attributes into the editor content.
Cancel	Closes the dialog, ignoring any data you have entered.

The Custom Tag Attributes Dialog

The Custom Tag Attributes dialog box (illustrated below) lets a user or developer view and apply or edit attributes to a custom tag.

This dialog is available on the context menu if the `cmdcusttagattrs` command exists in the configuration data.



The User Interface of the Custom Tag Attributes Dialog

NOTE The attribute values of the `tagattrdlg` element may suppress some of the buttons listed below. For more information, see "tagattrdlg" on page 702.

Field or button	Description
Used Attributes	Attributes assigned to the selected tag appear. You can <ul style="list-style-type: none"> • delete an attribute (using the Remove--> button) • insert a new attribute (using the <--Add button) • change its value (if one exists) • enter a value (if one does not exist)
New	Creates a new attribute for the selected tag. There is no validation that the attribute can be inserted into the tag definition. If you click this button, a new line appears at the top of the Attributes column for the new attribute. You can then move to the Value field and enter value for the new attribute if desired.
Remove	Removes the selected attribute.
Add	Adds the selected attribute.
Value	If a value was assigned to the selected attribute, it appears. You can leave it, delete it, or modify it as needed.
OK	Completes actions performed on screen.
Cancel	Closes the dialog, ignoring any data you have entered.

For more information, see "The Custom Tag Attributes Dialog" in the **eWebEditPro+XML** User Guide.

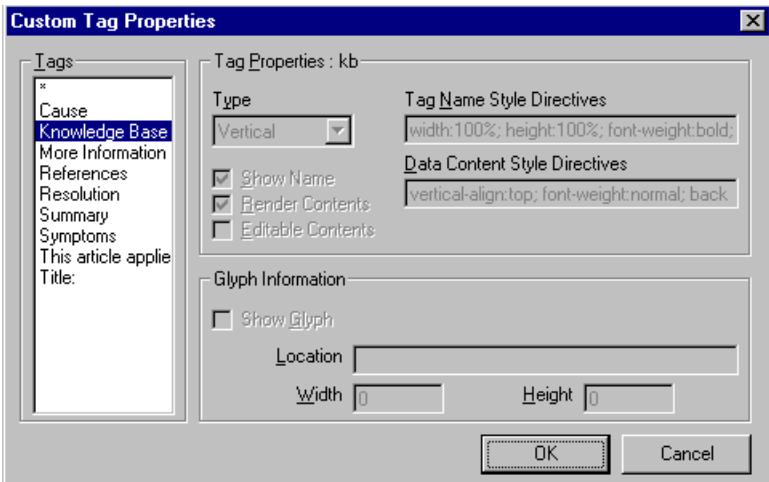
The Custom Tag Properties Dialog

The Custom Tag Properties dialog box (illustrated below) lets a user or developer view and add or modify the properties of a custom tag.

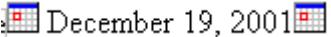
This dialog is available on the context menu if the `cmdcusttagprop` command exists in the configuration data.

The User Interface of the Custom Tag Properties Dialog

NOTE The attribute values of the `tagpropdlg` element may change the functionality of this dialog. For more information, see "tagpropdlg" on page 703.



Button or Field	Description
Tags	Display all available tags. If you selected a tag before launching this dialog box, that tag is highlighted.
Type	The type of tag being inserted or modified. See "Types of XML Tags" on page 686.
Show Name	Check this box to display the tag's caption as part of the tag's appearance on the screen.
Render Contents	Currently not implemented
Editable Contents	Currently not implemented
Tag Name Style Directives	The style definition for the tag section of an attribute. For more information, see "style" on page 685.
Data Content Style Directives	The style definition for the data section of an attribute. For more information, see "dstyle" on page 685.

Button or Field	Description
Show Glyph	<p>Check this box if you want a glyph (also known as an icon) to appear before and after your text (see illustration below).</p>  <p>If you check this box, enter the glyph's location, height, and width information in the subsequent fields.</p>
Location	The URL of the file to display as the glyph. This should be a location over the internet or intranet.
Width	The width in pixels at which to display the glyph. To display the glyph at its actual size, enter 0.
Height	The height in pixels at which to display the glyph. To display the glyph at its actual size, enter 0.
OK	Completes actions performed on screen.
Cancel	Closes the dialog, ignoring any data you have entered.

XML Objects

Two XML objects are offered to the user:

- [XML Object Interface](#)
- [XML Data](#)

This section introduces those objects.

XML Object Interface

The XML Object Interface is retrieved using the XMLProcessor method in the editor. Here is an example.

```
function LoadSchema(sEditorName, sSchemaPath, sNSTarget)
{
    var objXmlDoc = eWebEditPro.instances[sEditorName].editor.XMLProcessor();
    var sSchemaPath = document.frmeditor1.schemaFilePath.value;
    var sNSTarget = document.frmeditor1.namespacetarget.value;
    objXmlDoc.AddSchema(sSchemaPath, sNSTarget);
    if(0 != objXmlDoc.getPropertyInteger("ErrorCode"))
    {
        alert("Error: " + objXmlDoc.getPropertyString("ErrorReason"));
    }
}
```

For more information on this object, see "["XML Object"](#)" on page 18.

XML Data

The XML Data object contains functionality to retrieve and affect selected tags in the content. This object is retrieved using the XMLFacia ActiveTag method.

All methods and properties pertain only and directly to the element returned from the ActiveTag method. Here is an example:

```
function ShowElementLimitations(sEditorName)
{
    var objElem = eWebEditPro.instances[sEditorName].editor.XMLProcessor().ActiveTag();
    var iLim = 0;

    if(false == objElem.IsValid())
    {
        alert("No valid tag is selected.");
    }
    else
    {
        alert("Min Element Count: " + objElem.ElementMinCount());
        iLim = objElem.ElementMaxCount();
        if(-1 == iLim)
        {
            alert("Max Element Count: Unlimited");
        }
    }
}
```

```
    }
    else
    {
        alert("Max Element Count: " + iLim);
    }
```

For more information on this object, see "XML Data Object" on page 23

Best Practices for Using Custom Tags

This chapter provides tips and suggestions for developers who want to implement **eWebEditPro+XML**. It covers the following topics.

- Ways to assign tag definitions
- User Selectable Tag Looks
- Ideas for appearance
- Browser limitations
- Tag type considerations
- Avoid use of namespace with HTML tags

Ways to Assign Tag Definitions

Defining tags is very dynamic. Tag definitions can be changed on the fly by client scripting, or remain static in the configuration data. Also, when tags are assigned, they can either be added to or replace the existing set of tags.

Here are the ways that you can define XML tags.

- Not at all
- Within the configuration XML data

Not Defining Any Tags

The easiest way to handle the definition of XML tags is to not define them. Instead, let the default configuration define tag information. You can use any number of tags without defining them.

To do this, enter values for all attributes of the tagdefault element (see “[Tagdefault Element](#)” on page 693). You can use the sample installed with **eWebEditPro+XML** 2.5 or higher, but the default configuration may not meet your requirements. In other words, you probably will change the look of the tags.

Below are a few approaches you can take when defining default tag appearance in the editor.

All Tags are Invisible

No tags are visible with this configuration.

```
<tagdefault type="nonempty" visible="false"
    style="" dstyle="" astyle="">
    <glyph visible="false"/>
</tagdefault>
```

If you use this configuration, the user does not know that the tags exist in the content.

Because the `visible` attribute is set to **false**, the user does not see the tag's description. The empty `style` and `dstyle` attributes result in no special formatting to distinguish the tags from surrounding text. Also, the `glyph` element's `visible` attribute must be set to **false**.

The only drawback to this configuration is that tags cannot enclose large content that contains standard blocking tags, such as `<p>` or `<table>`. This is because the tag is closed before the blocking tag is closed.

See Also: "Blocking vs. Non-Blocking" on page 725

If your tags involve blocking tags, change the nonempty type to vertical or horizontal, as illustrated below.

```
<tagdefault type="vertical" visible="false"
    style="" dstyle="" astyle=""
    <glyph visible="false"/>
</tagdefault>
```

The difference between the two definitions is that a line break will appear between where the tag starts and ends.

All Tags Define Sections of Content

This configuration is a substitute for placing multiple editors on a page. Each tag's data and tag section clearly indicate the tag and where to enter content.

```
<tagdefault type="vertical"
    style="font-size:large; font-family:arial; font-weight:bold;
        color:yellow; background-color:blue; border:solid blue 1px; margin:2px; width:99%
    dstyle="background-color:white; color:black; border:solid black 1px; padding:2px;
        width:90%">
    astyle="font-size:normal;font-weight:normal" ashown="true">
        <glyph visible="false"/>
</tagdefault>
```

If a user can insert blocking tags, you must create JavaScript to ensure that the tags are brought applied correctly.

Defining in the Configuration XML Data

The best way to quickly assign unique characteristics to tags is to define them in the configuration XML data. This is a good choice for your first use of XML tags.

See Also: "Defining Tags in Config XML" on page 648

User Selectable Tag Looks

You can define a list of tag looks. Users pick a look from the list, and the client script assigns the look's definitions.

```
function useTagLook(sEditorName, iUseLook)
{
    var objEditor = eWebEditPro.instances[sEditorName];
    objEditor.editor.ExecCommand("cmddefinetags", g_TagLook[iUseLook], 0);
}
```

If the user changes the look of an existing tag, a cleaning must occur to replace the look.

Defining at Load Time

You can define external tags at load time. This is done when the editor signals that it is ready for interactions from the external client.

To define external tags at load time, define the eWebEditProReady JavaScript function, which the editor calls when it is ready.

See Also: "Event: eWebEditProReady" on page 191

NOTE This function does not exist in the standard JavaScript installed with **eWebEditPro+XML**.

```
function eWebEditProReady(sEditorName)
{
    useTagLook(sEditorName, 0);
}
```

If you assign tag looks using this function, be sure it is the only action performed in this function. Because JavaScripting is asynchronous, any commands sent to the editor after this may not execute.

NOTE The method of assigning tag definitions is very flexible. But, when a page refresh occurs, tags use the default tag settings until the definitions are assigned. This time period can last several seconds, depending on the user's environment.

Assigning External XML Stream

The most dynamic way to define tags externally is to build an XML stream that matches the data contained in an XML file and assign that stream to the editor. An example is below.

```
function eWebEditProReady(sEditorName)
{
    var objEditor = eWebEditPro.instances[sEditorName];
    AssembleTagDefinition();
    objEditor.editor.ExecCommand("cmddefinetags", g_TagLook, 0);
}
```

Ideas for Appearance

Use of Color

Use color to guide the user to a tag's content area. Light colors surrounded by dark colors draw a user's attention for content entry.

As an example, define a tag with a black border and caption background, and a white background for the data area.

```
<tagspec name="NewsItem" type="vertical" visible="true"
style="font-weight:bold;background-color:black;color:white; border:solid black 1pt;
margin:3pt; width:90%"
```

```
dstyle="background-color:white; color:black; margin:3pt">
  <caption>List of Topics</caption>
</tagspec>
```

Defining a border around the data area (using the `dstyle` attribute) reinforces the impression that the area is for content entry.

```
dstyle="background-color:white; color:black; margin:3pt; border:solid black 1pt"
```

Data Width

A horizontal tag's caption expands to most of the width of a tag section. To match the size of the tag section with the data section, set the width of the `dstyle` attribute to 100%.



```
<tagspec name="CopyrightDate" type="horizontal"
  style="background-color:silver; border:solid blue 1pt"
  dstyle="background-color:white; color:black; width:100%">
  <caption>Contributor</caption>
  <glyph src="[eWebEditProPath]/btnhy.gif" visible="false" width="0" height="0"/>
</tagspec>
```

Setting the width to a percentage adjusts the caption area to the specified size. If you do this for a group of tags, they all have the same size caption area.

```
<tagspec name="CopyrightDate" type="horizontal"
  style="background-color:silver; border:solid blue 1pt"
  dstyle="background-color:white; color:black; width:80%">
  <caption>Contributor</caption>
  <glyph src="[eWebEditProPath]/btnhy.gif" visible="false" width="0" height="0"/>
  <simtaglist name="deftaglist">
    <simtag name="CopyrightLine"></simtag>
    <simtag name="Copyright"></simtag>
    <simtag name="CopyrightDate"></simtag>
    <simtag name="CopyrightHolder"></simtag>
    <simtag name="DateId"></simtag>
  </simtaglist>
</tagspec>
```

Browser Limitations

The editor uses functionality provided by the browser to display tags to the user.

If the user is working with a version of Internet Explorer prior to 5.5, the user is able to edit the tag's caption. However, when a cleaning is performed, the edits are lost and the caption is restored. This problem does not occur with Internet Explorer 5.5 and up.

If the user works with Netscape, Internet Explorer still must be installed on the client. The installed IE version determines whether or not the user will have the problem.

Tag Type Considerations

There are considerations when selecting the display type for a tag.

Look

Use horizontal and vertical tag types for data entry locations. They effectively block off data, isolating it from surrounding information.

Non-empty and empty tags are normally applied to items within a content block. Typically, a user enters content and then selects items within the content to mark with tags.

Blocking vs. Non-Blocking

Horizontal and vertical tag types are seen as blocking tags by the editor. (Blocking tags are similar to the `<p>` and `<table>` HTML blocking tags.) The nonempty and empty tag types are *not* blocking tags.

If non-blocking tags are placed around blocking tags, the non-blocking tags may be removed. This occurs because the editor terminates the non-blocking tag before the blocking tag. If the result is no content within the shortened tag, the tag is removed since nonempty tags with no content are automatically removed.

Non-Empty Tag Problem

A non-empty tag with icons may create overlapping tags. This can occur because these tags allow the user to move the beginning and ending tags. However, if a user moves a tag within other tags, such as bold or font tags, overlapping tags may result.

This is only a concern when icons are visible in non-empty tags.

Determining Content Size

The `EstimateContentSize` method does not return the correct size of an XML document. This occurs because processing the tags for display artificially increases the document size calculated by the method.

See Also: "[Method: EstimateContentSize](#)" on page 79

Use Tag Properties Dialog as Diagnostic Tool

It is best to use the Custom Tag Properties dialog box as a diagnostic tool for tags, rather than as a tool for the end user. There are two reasons for limiting the dialog's use in this way.

- Many dialog items, such as the style values, are confusing to a novice user.
- Properties modified in the dialog are lost when the page is refreshed. All tags either revert to the tag's definition or use the default settings.

Don't Use a Namespace with HTML Tags

Namespaces used with the standard HTML tags are not recognized by the editor. If a namespace is used with the HTML tags, each tag is assumed to be a custom tag and displayed as a custom tag.

Validating XHTML Content

Ektron CMS400.NET can check content for two types of compliance:

- XHTML standards - see "["Validating Content for Compliance with XHTML Standards" on page 732](#)
- accessibility standards established by agencies such as W3C Markup Validation Service - see "["Validating Content For Compliance with Accessibility Standards" on page 727](#)

This check is performed when a user tries to save, check in or (submit for) publish content.

In addition, some dialogs add accessibility data to **Ektron CMS400.NET** content. As examples, the Picture Properties is used to add an image's <alt> attribute. When a user clicks **OK** to save these dialogs, **Ektron CMS400.NET** can check that compliance information is being added. See "["Dialogs that Allow Input of Accessibility Information" on page 731](#)

This section explains how to enable these features, and how to customize them to meet your needs.

The rest of this section provides details about validating XHTML content through these topics.

- "["Validating Content For Compliance with Accessibility Standards" on page 727](#)
- "["Dialogs that Allow Input of Accessibility Information" on page 731](#)
- "["Validating Content for Compliance with XHTML Standards" on page 732](#)

Validating Content For Compliance with Accessibility Standards

To have **Ektron CMS400.NET** check content for compliance with accessibility standards, enable accessibility *and* define an XSLT or Web site to check against.

Enabling Accessibility

In Ektron CMS400 .NET, accessibility options appear on the Configuration Setup screen under **Accessibility/Section 508 Evaluation** (below the **Editor Options**). The options are explained below.

- **Do not validate** - do not check content for compliance with accessibility standards
- **Warn if fails** - the user is warned but allowed to save content
- **Enforce** - the user must bring the content into compliance before submitting it for publication or publishing it; however, can save it or check it in after being warned.

Defining the XSLT or Web Site

When a user tries to save content and **Accessibility/Section 508 Evaluation** is set to **Warn if Fails** or **Enforce**, the content is checked against an XSLT, a Web site designed to validate content, or both.

If you enable both, the content is checked in this order.

1. XSLT
2. Web site

If the first check fails, the second check is not performed.

The XSLT File

Ektron CMS400.NET content can be checked against an XSLT that enforces W3C's WCAG Level 1 requirement. This XSLT, `ektaccesseval.xslt`, resides in the folder to which **Ektron CMS400.NET** is installed.

The XSLT is identified in the following line (highlighted in red) of the configuration data. (In **Ektron CMS400.NET**, the configuration data is stored in site root folder\Workarea\ewebeditpro\CMS_config.aspx.)

```
<!-- accessibility options are "strict", "loose" and "none" -->
<validate accessibility="none" suggestdefaultval="true">
<xslt enabled="true" src="[eWebEditProPath]/ektaccesseval.xslt" />
```

You can modify this XSLT as needed.

Or, you can place a different XSLT in the **Ektron CMS400.NET** root directory and refer to it in the `src` attribute shown above. If you use a different XSLT and validation fails, it should return a description of the problem(s) as HTML. If validation succeeds, it should return nothing.

Validating Against a Web Site

You can also validate XHTML content against a Web site that evaluates its compliance with accessibility standards. Two such sites are listed in the configuration data:

- hermish.com
- webaim.org

You can use either listed site or any other site designed to check for accessibility compliance. Regardless of the site's origin, you can only use one.

The Web sites are specified in these lines of the configuration data. To activate a site, set `enabled` to `true`.

```
<online enabled="false" type="string" contentname="myHTML" src="http://www.hermish.com/
check_this.cfm">
<data name="URLtest">Your HTML</data>
<data name="CheckURL">1</data>
<data name="p1">1</data>
<data name="p2">1</data>
<data name="p3">1</data>
<data name="s508">1</data>
</online>
```

```
<online enabled="false" type="file" contentname="upload" src="http://wave.webaim.org/wave/
Output.jsp">
    <data name="Submit">Submit</data>
    <data name="IPAddress">127.0.0.1</data>
</online>
```

Enabling a Site Not Included in the Configuration Data

Ektron only supports sites listed in the configuration data by default. If you want to validate content against another Web site designed to do that, follow these guidelines.

- Web sites that perform validation typically accept input in this format.

```
<form action="http://www.testing.com/validationaction.asp">
    <textarea name="testcontent"></textarea>
    <input type="submit" name="btnSubmit" value="Validate the
content" />
</form>
```

To accommodate this format, define the configuration data's `<online>` element as follows.

```
<online enabled="true" type="string" contentname="testcontent"
src="http://www.testing.com/ validationaction.asp"
keywordsearchresult="failure" keyword="invalid">
    <data name=" btnSubmit"> Validate the content</data>
</online>
```

The following table describes `<online>`'s attributes.

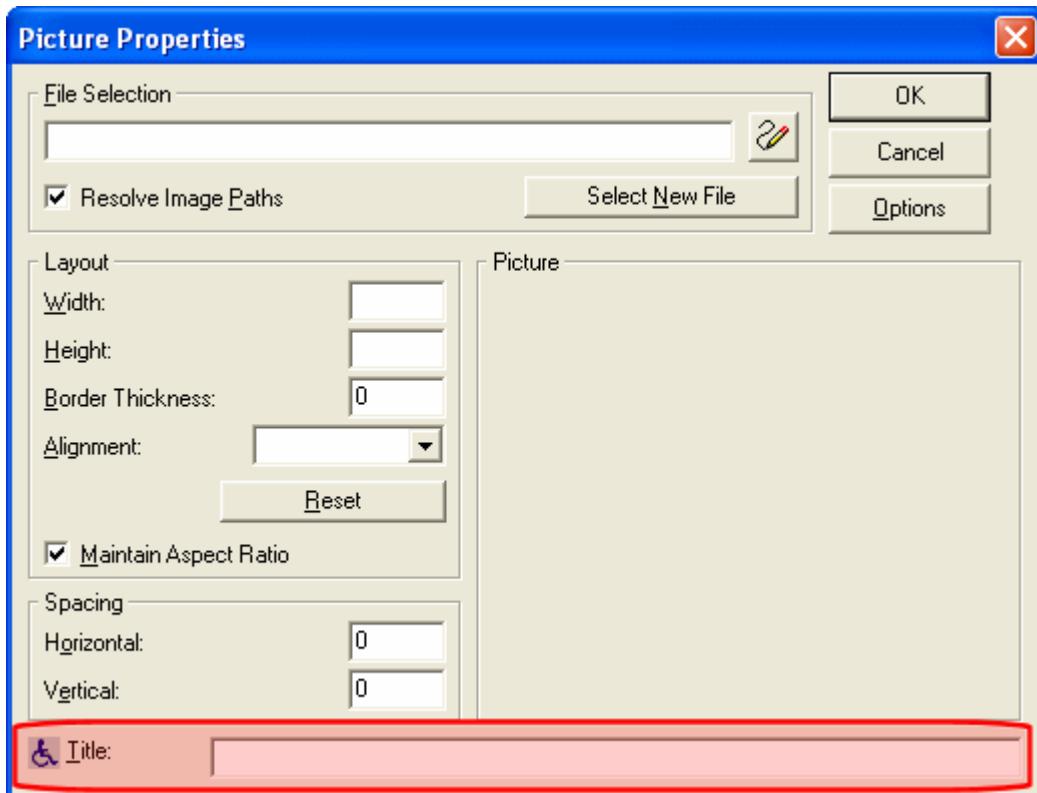
<Online> Attribute	Description
type	<p>If the form accepts an HTML string as input, enter string. If the form accepts a file name as input, enter file.</p>
contentname	<p>Enter the value of the <code>name</code> attribute of the <code>textarea</code> or <code>input</code> element in the form. For a form that accepts an HTML string, the <code>textarea</code> may appear as:</p> <pre><textarea name="testcontent"></textarea></pre> <p>In this case, the value for contentname would be testcontent.</p> <p>For a form that accepts a file, the <code>input</code> element may appear as:</p> <pre><input type="file" name="testfile" /></pre> <p>In this case, the value for contentname would be testfile.</p>

<Online> Attribute	Description
src	<p>To obtain the URL of the Web page that validates HTML content, go to the site and view the source. From there, you should be able to obtain the form's action attribute. Place that into this attribute.</p>
keywordsearchresult	<p>Takes one of two values: <code>failure</code> or <code>success</code>.</p> <ul style="list-style-type: none"> • If set to <code>success</code> and Ektron CMS400.NET finds the keyword (set in the keyword attribute) in the validating Web page's response, the content is saved. • If set to <code>success</code> and Ektron CMS400.NET does <i>not</i> find the keyword in the validating Web page's response, the Web site's failure page appears. Typically, the page explains why the content failed validation. • If set to <code>failure</code>, and Ektron CMS400.NET finds the keyword (set in the keyword attribute) in the validating Web page's response, Web site's failure page appears. Typically, the page explains why the content failed validation. • If set to <code>failure</code>, and Ektron CMS400.NET does not find the keyword in the validating Web page's response, the content is saved. <p>If you do not define a keyword and keywordsearchresult, the validating Web page's response (or report) appears.</p>
keyword	<p>Enter a text string that Ektron CMS400.NET searches for in the source of the Web site's response to the validation check.</p> <p>For example, if the site displays Congratulations when validation passes, enter Congratulations in this attribute. In this case, enter <code>success</code> as the keywordsearchresult.</p> <p>As another example, if you know that a Web site displays the <code>images/iconViolation.gif</code> image when validation fails, enter that as the keyword. In this case, enter <code>failure</code> as the keywordsearchresult.</p>

<Online> Attribute	Description
data	<p>Enter additional form fields to be posted to the Web site during form submission.</p> <ul style="list-style-type: none"> The name attribute of the data tag specifies the name of the form field. The value between the data tags is the value of the form field when submitted. <p>You may specify as many data tags as needed. For example</p> <pre><data name="btnSubmit">Validate the content</data></pre>

Dialogs that Allow Input of Accessibility Information

Several dialog boxes have fields that collect required accessibility data. An example of such a field is highlighted below.



When the configuration data's `suggestdefaultval` property is set to `true`, **Ektron CMS400.NET** tries to ensure that a value exists in required accessibility fields of the following dialogs.

Dialog	Field	How filled if no value exists
Picture Properties	Title	Contents of File Selection field copied
Hyperlink	Title	Contents of Text field copied
Group Box	Caption	Contents of Descriptive Name field copied to Caption field
Several Data Designer dialogs, such as Plain Text Field, Image Only field, Link Field, Select List Field	Tooltip text	Contents of Descriptive Name field copied

Validating Content for Compliance with XHTML Standards

Ektron CMS400.NET content can be validated against the XHTML 1.0 schema, a Web site, or both. These are described in "The Schema" on page 732 and "The Web Site" on page 733.

NOTE

This validation occurs when content is saved regardless of the value of the `<accessibility>` attribute.

Besides specifying a schema or Web site, you need to uncomment the following line in `ektfilter.xslt` (located in the `webroot/ewebeditpro5` directory). This XSLT removes non-W3C-compliant tags and attributes.

```
<xsl:include href="ektfilterxhtml110.xslt"/>
```

The Schema

Two schemas from the W3C site are provided with **Ektron CMS400.NET**.

- www.w3.org/2002/08/xhtml/xhtml1-strict.xsd
- www.w3.org/2002/08/xhtml/xhtml1-transitional.xsd

The schemas are installed to the **Ektron CMS400.NET** root folder. You choose a schema in the following lines of the configuration data.

```
<schema enabled="false" src="[eWebEditProPath]/xhtml1-strict.xsd"/>
<schema enabled="false" src="[eWebEditProPath]/xhtml1-transitional.xsd"/>
```

To specify a schema against which to validate content, set `enabled` to `true`.

The options are explained below.

- **Both schemas disabled**- do not check content for compliance with XHTML standards
- **transitional or strict enabled** - the user must bring the content into compliance before submitting it for publication or publishing it; however, can save it or check it in after being warned.

You can modify the .xsd file as needed. Or, you can place a different .xsd file in the eWebEditPro+XML root directory and refer to it in the `src` attribute shown above.

IMPORTANT!

Because of an error in XML parser 4 (see <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q262585>), Ektron has removed the namespace from the schemas. If you design your own schema, do not use 'xml' to begin user-defined namespace prefixes.

The Web Site

You can validate content against any Web site that enforces W3C's XHTML standards. One site is listed in the configuration data: `validator.w3.org`.

You can use that site or another site designed to check for XHTML compliance.

To activate the site in the configuration data, set `enabled` to `true` in the code snippet shown below.

```
<online enabled="false" type="string" contentname="fragment" src="http://validator.w3.org/check"/>
```

If you want to use a different Web site, follow the directions in "Enabling a Site Not Included in the Configuration Data" on page 729.

Integrating eWebEditPro+XML

NOTE

If **eWebEditPro+XML** is used on a platform (that is, a browser or operating system) that does not support **eWebEditPro+XML**, a textarea field automatically appears in its place. No extra work is required to handle unsupported platforms.

This section explains how to integrate **eWebEditPro+XML** in the following environments.

- ASP.NET
- ASP
- ColdFusion
- JSP
- PHP

Regardless of your server environment, you can always integrate **eWebEditPro+XML** using JavaScript. “[Integrating eWebEditPro+XML Using JavaScript](#)” on page 766 explains how to do that.

Each section provides step-by-step instructions for integrating **eWebEditPro+XML** in that environment. Documentation is also provided for the samples supplied.

Integrating eWebEditPro+XML with JSP

Using the Sample Pages

When you download **eWebEditPro+XML**, Ektron provides sample pages that include the editor. The default location is `www.mywebsite.com/ewebeditpro5/samples/jsp`.

You should copy these samples to another directory or rename them, and then modify them as needed. If you do not copy them, any changes you make could be overwritten when you reinstall or upgrade **eWebEditPro+XML**.

Creating Your Own Page

If you want to create a new JSP page and place **eWebEditPro+XML** on that page, the page needs to include these actions.

1. Include the `ewebeditpro.jsp` file.
2. Set up a form.
3. Place the editor on the form.
 - Retrieving and Loading Data Designer Content
4. Add a submit button.

The rest of this section explains how to complete these tasks.

Including a Reference to `ewebeditpro.jsp`

Your JSP page must contain an `include` command that specifies a relative path to the `ewebeditpro.jsp` file. Place the `include` line within the page's head tags.

Use this syntax to indicate a relative path to the domain name of this file.

```
<head>
<%@ include file="/ewebeditpro5/ewebeditpro.jsp" %>
</head>
```

Setting Up a Form

When setting up a form, follow these steps.

1. Declare a form.
2. Enter a url as the action. This defines the page that manipulates the user's input when the user presses the submit button.
3. Enter **Post** as the method.

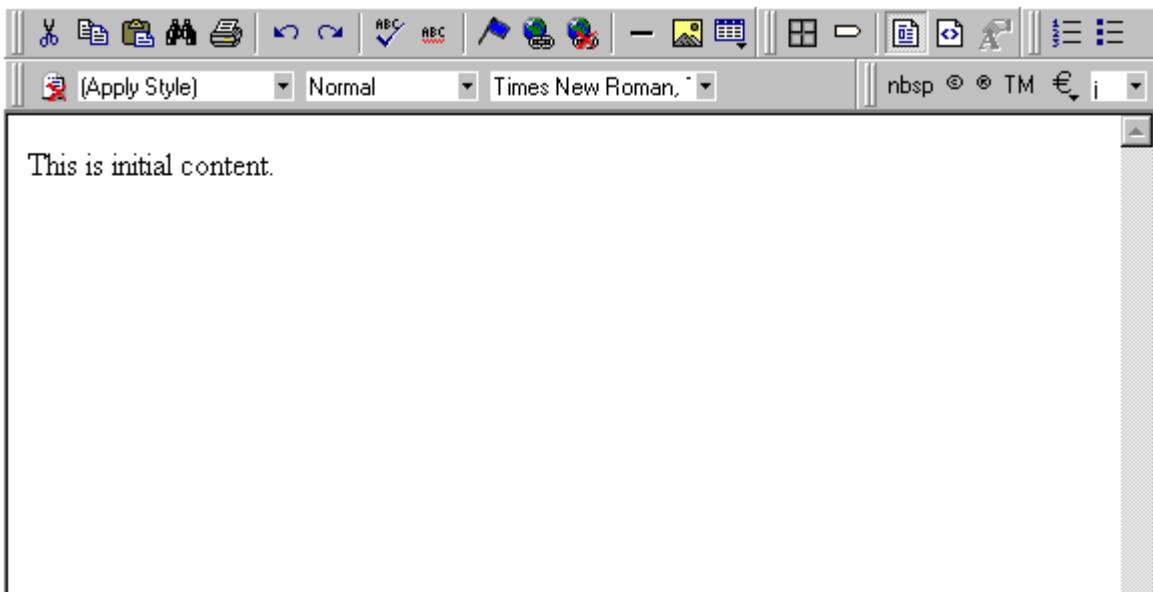
Here is a sample form declaration.

```
<form action="multiedit.jsp?preview" method="POST">
```

Placing the Editor on the Form

Place the editor within the form tags as a box whose width and height you specify.

This illustration depicts the editor appearing as a box.



For each editor that you want to place on the Web page, you

- change parameter values as needed
- insert the editor

Changing Parameter Values

If you want to change parameters that affect all instances of the editor, edit the ewebeditprodefaults.js file using a standard text editor. (For a description of each parameter, see “[Customizable JavaScript Files](#)” on page 291.)

To change the parameters only for the instance of **eWebEditPro+XML** that you are placing on the JSP page, enter the following code. In this example, you

remove the About button () from the toolbar.

```
<script language="JavaScript">
eWebEditPro.parameters.reset();
eWebEditPro.parameters.hideAboutButton="False";
</script>
```

If you are placing more than one editor on a page, and you want the parameters for each editor to be different, begin the parameter code with `eWebEditPro.parameters.reset()`. This line restores the parameters to the default values set in ewebeditprodefaults.js.

Inserting the Editor

To place the editor on a JSP page, enter a line with the following elements within the form.

```
<%= eWebEditProEditor("field name", width, height, initial content) %>
```

Argument	Description
field name	Enter the name of the field that stores content within quotes (""). It does not matter what the name is, but the field name on the page that retrieves the content must match this name.
width, height	<p>Enter the width and height of the editor in percent or pixels.</p> <ul style="list-style-type: none"> If a percent, enclose the value in quotes ("") and follow it with a percent sign (%), for example “50%”. If pixels, quotes are optional, for example, 500. <p>IMPORTANT! If you are integrating eWebEditPro+XML in a Java Server Page (JSP) environment, you <i>must</i> surround width and height values in pixels with quotes. For example:</p> <pre><%= eWebEditProEditor("TextHTML1", "100%", "250", strContent1)%></pre>
initial content	<p>If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text.</p> <ul style="list-style-type: none"> To enter text, enclose it within quotes (""). To enter a variable, define it elsewhere in the file.

Here is an example of a line that calls the editor. In this example, the initial content is defined in the variable `strContent1`, which is defined elsewhere in the file.

```
<%= eWebEditProEditor("TextHTML1", "100%", "250", strContent1) %>
```

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see ["Retrieving and Loading Data Designer Content" on page 607](#) and ["JSP" on page 611](#).

Adding a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server after entering it. Here is an example of a line that contains a submit button.

```
<input type="submit" name="btnSubmit" value="Preview">
```

Integrating eWebEditPro+XML with ASP

Using the Sample Pages

When you download **eWebEditPro+XML**, Ektron provides sample pages that include the editor. The pages are located below the folder to which you installed **eWebEditPro+XML**. The default location is
 C:\Inetpub\wwwroot\ewebeditpro5\samples\asp.

You should copy these samples to another directory or rename them, and then modify them as needed for your users. If you do not copy them, any changes you make could be overwritten when you reinstall or upgrade **eWebEditPro+XML**.

Creating Your Own Page

If you want to create a new ASP page and place **eWebEditPro+XML** on that page, the page needs to include these actions.

1. Include the ewebeditpro.asp file.
2. Set up a form.
3. Place the editor on the form.
 - Retrieving and Loading Data Designer Content
4. Add a submit button.

The rest of this section explains how to complete these tasks.

Including a Reference to ewebeditpro.asp

Your ASP page must include a reference to the ewebeditpro.asp file. You can use a relative or an absolute path. Ektron recommends using an absolute path.

You must place the #Include line within the page's head tags.

NOTE For a relative path, follow the include command with **file**. For an absolute path, follow the include command with **virtual**.

Entering a Relative Path

Use this syntax to indicate a *relative* path to the domain name of this file.

```
<head>
<!-- #Include file="../../ewebeditpro.asp" -->
</head>
```

Entering an Absolute Path

Use this syntax to indicate an *absolute* path to the domain name of this file.

```
<head>
<!-- #Include virtual="/ewebeditpro5/ewebeditpro.asp" -->
</head>
```

Setting Up a Form

When setting up a form, follow these steps.

1. Enter a URL as the action. This defines the page that manipulates the user's input when the user clicks the submit button.
2. Enter **Post** as the method.

Here is a sample form declaration.

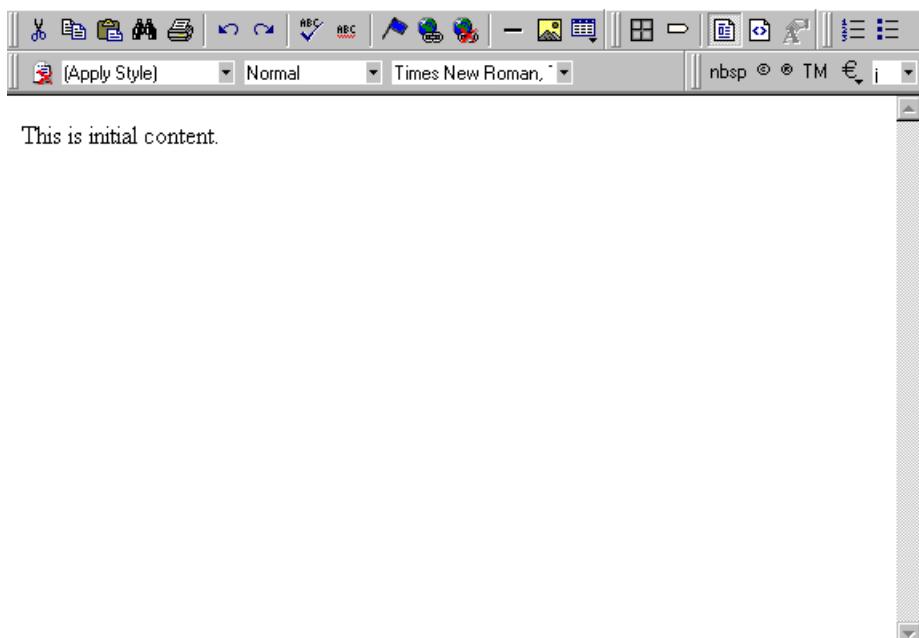
```
<form action="multiedit.asp?preview" method="POST">
```

Placing the Editor on the Form

Place the editor within the form tags. You can place the editor as a

- box whose width and height you specify, or
- button that, when clicked, displays a new screen with the editor

This illustration depicts the editor as a box.



This illustration depicts it as a button.



For each editor that you want to place on the web page, you

- change parameter values as needed
- insert the editor

Changing Parameter Values

If you want to change parameters that affect *all* instances of the editor, edit the ewebeditprodefaults.js file. You can use a standard text editor such as Notepad.

(For a description of each parameter, see “[Customizable JavaScript Files](#)” on [page 291](#).)

To change parameters only for the instance of **eWebEditPro+XML** you are placing on the page, enter the following code. In this example, you edit the parameters to display the About button () on the toolbar.

```
<script language="JavaScript">
eWebEditPro.parameters.reset();
eWebEditPro.parameters.hideAboutButton="False";
</script>
```

If you are placing more than one editor on a page, and you want the parameters for each editor to be different, begin the parameter code with `eWebEditPro.parameters.reset()`. This line restores the parameters to the default values set in ewebeditprodefaults.js.

Inserting the Editor as a Box

To place the editor as a box on an ASP page, enter a line with the following elements within the form.

```
<% =eWebEditProEditor("field name", width, height, initial content) %>
```

Argument	Description
field name	Enter the name of the field that stores content within quotes (" "). It does not matter what the name is, but the field name on the page that retrieves the content must match this name.

Argument	Description
width, height	<p>Enter the width and height of the editor in percent or pixels.</p> <ul style="list-style-type: none"> • If a percent, enclose the value in quotes ("") and follow it with a percent sign (%), for example “50%”. • If pixels, quotes are optional, for example, 500.
initial content	<p>If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text.</p> <ul style="list-style-type: none"> • To enter text, enclose it within quotes (""). • To enter a variable, define it elsewhere in the file.

Here is an example of a line that calls the editor. In this example, the initial content is defined in the variable strContent1, which is defined elsewhere in the file.

```
<% =eWebEditProEditor("TextHTML1", "100%", 250, strContent1) %>
```

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see ["Retrieving and Loading Data Designer Content" on page 607](#) and ["ASP" on page 608](#).

Inserting the Editor as a Button

To place the editor as a button on an ASP page, enter within the form

- a field into which the user enters the content
- the button

Entering a Field

Enter a text area box, a text input field, or a hidden field that submits the content to the database.

Here is a typical text area field.

```
<textarea name ="text1" rows=20 cols=120>
sample text
</textarea>
```

Here is a typical text input field declaration, preceded by text that instructs the user what to do.

```
Enter some text: <input type="text" size=70 name="text1" value="Sample text">
```

Entering the Button

To add the button to the page, enter a line with the following elements.

```
<% =eWebEditProPopupButton( "button name", "text field name" ) %>
```

Argument	Description
button name	Assign the button any name you wish.
field name	Enter the name of the field that stores content within quotes (""). The field name must match the field named in the text input field declaration.

Here is a typical button declaration.

```
<% =eWebEditProPopupButton( "btnEditText1", "text1" ) %>
```

NOTE

To edit the button text, open the ewebeditpromessages.js file using a standard text editor such as Notepad. Within that file, edit the text within quotes that follows **popupButtonCaption**:

Adding a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server. Here is an example of a line that contains a submit button.

```
<input type="submit" name="btnSubmit" value="Preview">
```

Integrating eWebEditPro+XML with ASP.NET

Using the Sample Pages

When you download **eWebEditPro+XML**, Ektron provides sample pages that include the editor. The pages are located below the folder to which you installed **eWebEditPro+XML**. The default location is
 C:\Inetpub\wwwroot\ewebeditpro5\samples\aspnet.

You should copy these samples to another directory or rename them, and then modify them as needed for your users. If you do not copy them, any changes you make could be overwritten when you reinstall or upgrade **eWebEditPro+XML**.

Integrating eWebEditPro+XML on an ASP.NET Page

There are three ways to place **eWebEditPro+XML** on an ASP.NET page. Each technique is described below with its advantages. Ektron supplies sample code for each one.

Technique	Description	For more information, see
Using a function	<p>This is most similar to ASP programming. You call a Visual Basic function from your ASPX page.</p> <p>If you are migrating from ASP to ASP.NET and want to get it running quickly, you may want to start with this approach.</p>	“Using a Function” on page 745
Using a custom user control	<p>You add a custom tag to your ASPX page as you might any HTML control in ASP.NET.</p> <p>Although simple, this technique does not let you fully separate your code from the page's presentation and layout. Use this method if you do not need to use the code-behind concept.</p>	“Using a Custom User Control” on page 746
Using a custom server control	<p>This is the most complex. When using it, you must reference the eWebEditPro+XML server control in your ASP.NET project in VisualStudio.NET.</p> <p>The server control supports the code-behind concept that lets you separate your code from the page's layout and presentation. Use this technique if you want to use the code-behind concept. It can be used with Visual Basic, C#, or any other .NET language.</p>	“Using a Custom Server Control” on page 748

Using a Function

Ektron provides sample code to simplify the integration of **eWebEditPro+XML** with Microsoft ASP.NET. You can insert **eWebEditPro+XML** into an ASP.NET page just as easily as you can insert a text area field into an HTML page.

To insert **eWebEditPro+XML** into an ASP.NET page using the ASP function, follow these steps.

1. Include a reference to ewebeditpro.aspx.
2. Set up a form.
3. Place the editor on the form.
 - Retrieving and Loading Data Designer Content
4. Add a submit button.

Including a Reference to ewebeditpro.aspx

1. To include a reference to ewebeditpro.aspx, place the `#Include` line within the page's head tags. Ektron recommends using an absolute path. To indicate an absolute path, use this syntax:

```
<head>
<!-- #Include virtual="/ewebeditpro5/ewebeditpro.aspx" -->
</head>
```

Setting up a Form

Here is a sample form declaration. (Be sure to enter `post` as the method.)

```
<form id="Form1" method="post" runat="server">
```

Placing the Editor on the Form

To place the editor on an ASP page, enter a line with the following elements within the form tags:

```
<% =eWebEditProEditor("field name", width, height, initial content) %>
```

Argument	Description
Field name	Enter the name of the field that stores content within quotes (""). It does not matter what the name is, but the field name on the page that retrieves the content must match this name.
Width, Height	Enter the width and height of the editor in percent or pixels. If a percent, enclose the value in quotes ("") and follow it with a percent sign (%), for example "50%". If pixels, quotes are optional, for example, 500.

Argument	Description
Initial content	If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text. To enter text, enclose it within quotes (""). To enter a variable, define it elsewhere in the file.

Here is an example of a line that calls the editor: (In this example, the initial content is defined in the variable strContent1, found elsewhere in the file).

```
<% =eWebEditProEditor("TextHTML1", "100%", 250, strContent1) %>
```

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see "Retrieving and Loading Data Designer Content" on page 607 and "ASP.NET" on page 609.

Adding a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server after entering it. Here is an example of a line that contains a submit button:

```
<input type="submit" name="btnSubmit" value="Preview">
```

or, add an ASP.NET button

```
<asp:Button id="btnSubmit" runat="server" Text= "Preview"></asp:Button>
```

Using a Custom User Control

If you want to create a new ASP.NET page and place **eWebEditPro+XML** on that page as a custom user control, follow these steps.

1. Register the control file, ewebeditpro.ascx.
2. Set up a form.
3. Place the editor on the form.
4. Add a submit button.
5. Gain access to the posted content.

The rest of this section explains how to complete these tasks.

Register the Control File ewebeditpro.ascx

Your ASP.NET (aspx) page must register the ewebeditpro.ascx user control file. To accomplish this, insert the <%@ Register tag at the top of your aspx page.

```
<%@ Register TagPrefix="ewep" TagName="eWebEditProEditor" src="/ewebeditpro5/ewebeditpro.ascx" %>
```

- TagPrefix determines a unique namespace for the user control
- TagName is the unique name for the user control

- The src attribute is the virtual path to the user control, for example ".../.../ewebeditpro.ascx" or "/ewebeditpro5/ewebeditpro.ascx"

Setting up a Form

Here is a sample form declaration. Be sure to enter post as the method.

```
<form id="Form1" method="post" runat="server">
```

Place the Editor on the Form

Place the user control tag in the Web Form page just as you would an ordinary server control (including the runat="server" attribute). To place the editor on a Web Form, enter a line with the following elements within the form tags.

```
<ewep:eWebEditProEditor id="TextHTML1" runat="server" height="250" width="100%" Text="initialcontent"></ewep:eWebEditProEditor>
```

Attribute	Description
id	Enter a unique name for each editor, for example MyEditor1, MyEditor2.
width, height	Enter the width and height of the editor in percent or pixels. Enclose the value in quotes (""). If a percent, follow it with a percent sign (%), for example "50%". If pixels, just enclose the value in quotes, for example, "500".
Text	If you want text to appear in the editor the first time a user views it, enclose the text within quotes (""). To dynamically pass content, set the property using a server-side script. For example, TextHTML1.Text = "initial content", where TextHTML1 is the id of the user control. Here is an example. <pre><script language="VB" runat="server"> Sub Page_Load(Sender As Object, E As EventArgs) If (Page.IsPostBack) TextHTML1.Text = Request.Form("TextHTML1") TextHTML2.Text = Request.Form("TextHTML2") Else TextHTML1.Text = "<p>Initial content 1</p>" TextHTML2.Text = "<p>Initial content 2</p>" End If End Sub </script></pre>

Here is an example that inserts two editors.

```
<ewep:eWebEditProEditor id="TextHTML1" runat="server" height="250" width="100%" Text="Editor 1"></ewep:eWebEditProEditor>
```

```
<br>
```

```
<ewep:eWebEditProEditor id="TextHTML2" runat="server" height="250" width="750" Text="Editor 2"></ewep:eWebEditProEditor>
```

You can also set the content using `TextHTML1.Text = "Some content"` and `TextHTML2.Text = "Some other content"` in a `Page_Load` subroutine or some other server-side event.

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see ["Retrieving and Loading Data Designer Content" on page 607](#) and ["ASP.NET" on page 609](#).

Add a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server after entering it. Here is an example of a line that contains a submit button.

```
<input type="submit" name="btnSubmit" value="Preview">
```

Or, you can add an ASP.NET button.

```
<asp:Button id="btnSubmit" runat="server" Text= "Preview"></asp:Button>
```

Gaining Access to the Posted Content

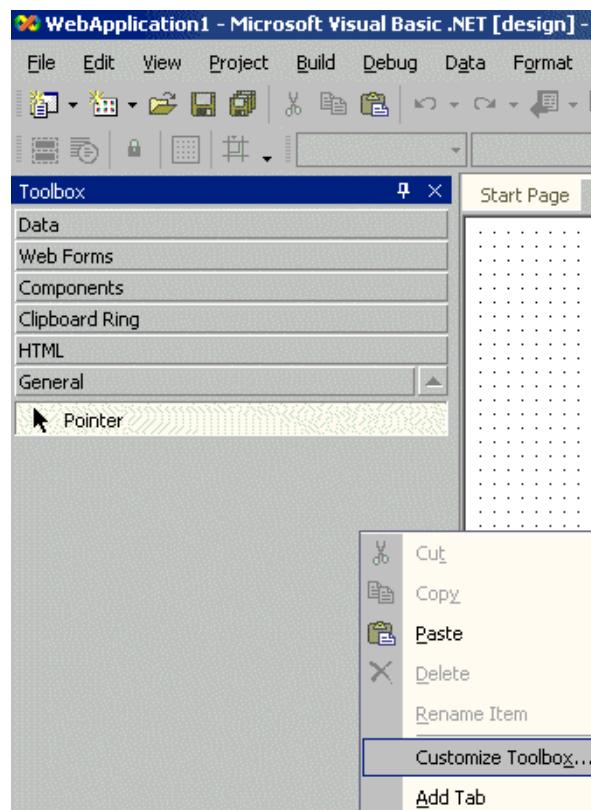
When posting to an action page, you have access to the posted content via `Request.Form("TextHTML1")`, where `TextHTML1` is the id of the user control.

```
<script language="VB" runat="server">
Sub Page_Load(Sender As Object, E As EventArgs)
If (Page.IsPostBack)
    TextHTML1.Text = Request.Form("TextHTML1")
    TextHTML2.Text = Request.Form("TextHTML2")
End If
End Sub
</script>
```

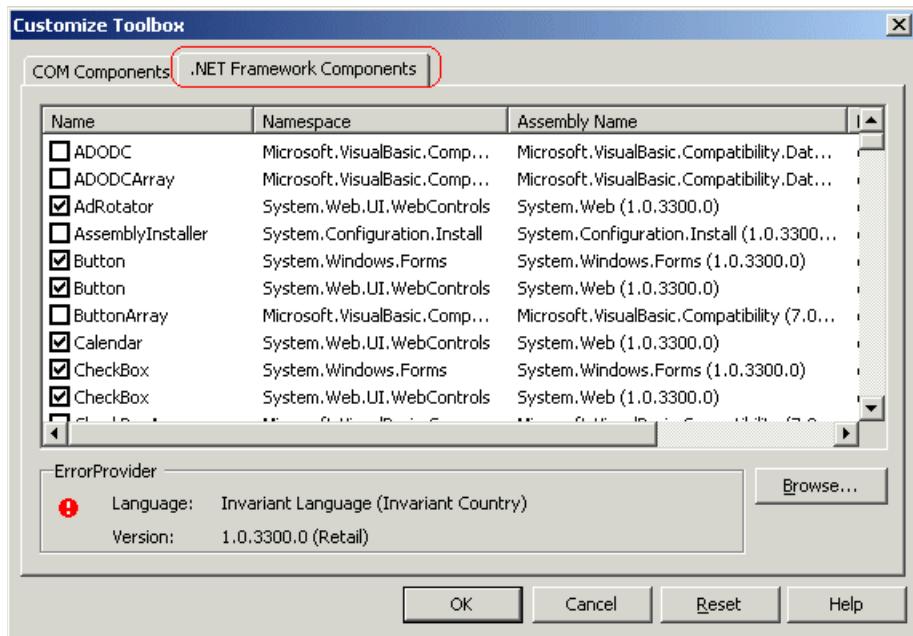
Using a Custom Server Control

If you want to create a new ASP.NET page and place **eWebEditPro+XML** on that page as a custom server control, follow these steps.

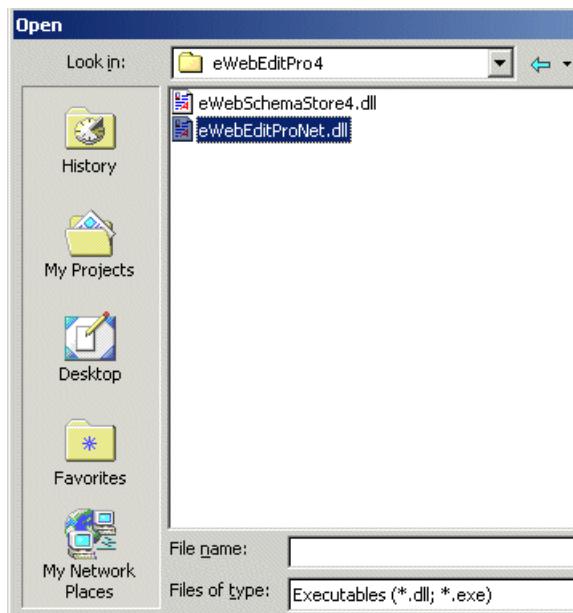
1. Open Microsoft Visual Studio.NET.
2. Select **Toolbox**, then right click the mouse and select **Customize Toolbox**. (See illustration below.)



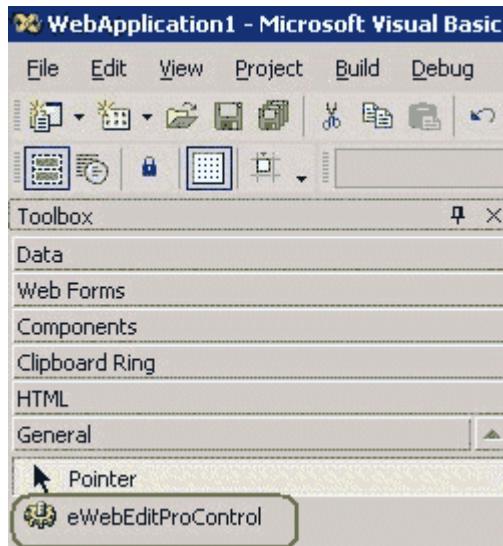
3. The Customize Toolbox dialog appears. Click **.NET Framework Components**. (See illustration below.)



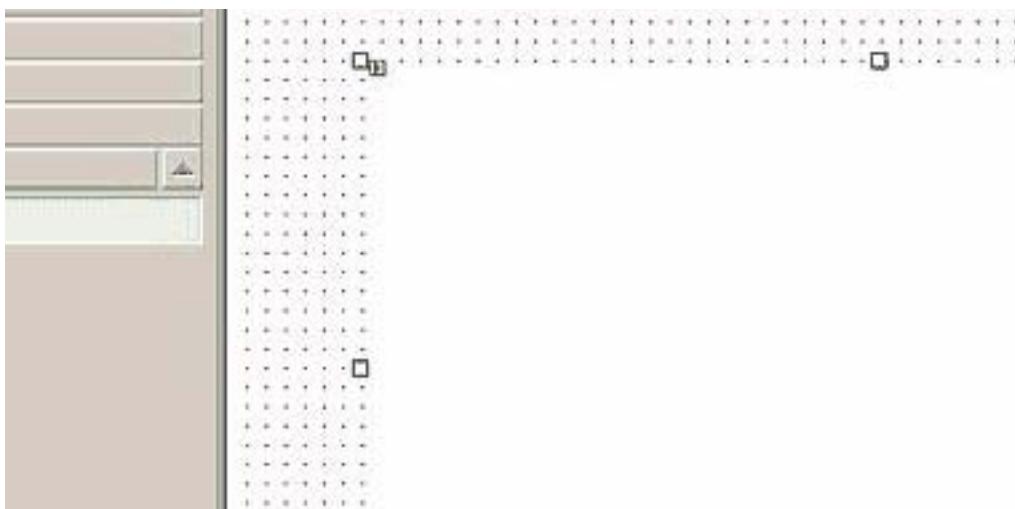
4. Browse to your program files directory then select Ektron/ewebeditpro5/eWebEditProNet.dll. (See illustration below.)



5. The file appears among the files in the Customize Toolbox dialog. Click OK.
6. Notice that the .dll file is now in the toolbox. (See illustration below.)



7. Drag and drop the **eWebEditPro+XML** control file into your form.
8. Enlarge the size of the control file. (See illustration below.)



9. Write the code-behind code to access the content. Here is an example.

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    EWebEditProControl1.Text = "This is an initial content.";
}
```

Validator Control Support

The server control supports validation using ASP.NET validation controls. The ASP.NET validation controls include RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, and CustomValidator.

Customizing eWebEditPro+XML Parameters

eWebEditPro+XML exposes the properties of the `eWebEditPro.parameters` object to the code-behind (for example, C#) on the server. As a result, you can set parameters using values known only on the server. The `parameters` object in the **eWebEditPro+XML** server control renders the JavaScript needed to set parameter values on the client side.

The **eWebEditPro+XML** placeholder control's properties appear in the Visual Studio .NET Properties dialog.

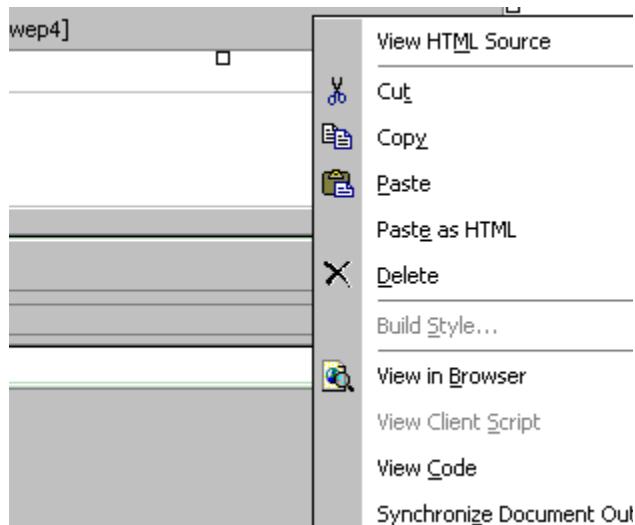
NOTE

Some properties and methods do not apply to **eWebEditPro+XML**. This is because the **eWebEditPro+XML** placeholder control is derived from `Microsoft.ContentManagement.WebControls.BasePlaceholderControl`, which is derived from the ASP.NET WebControl class.

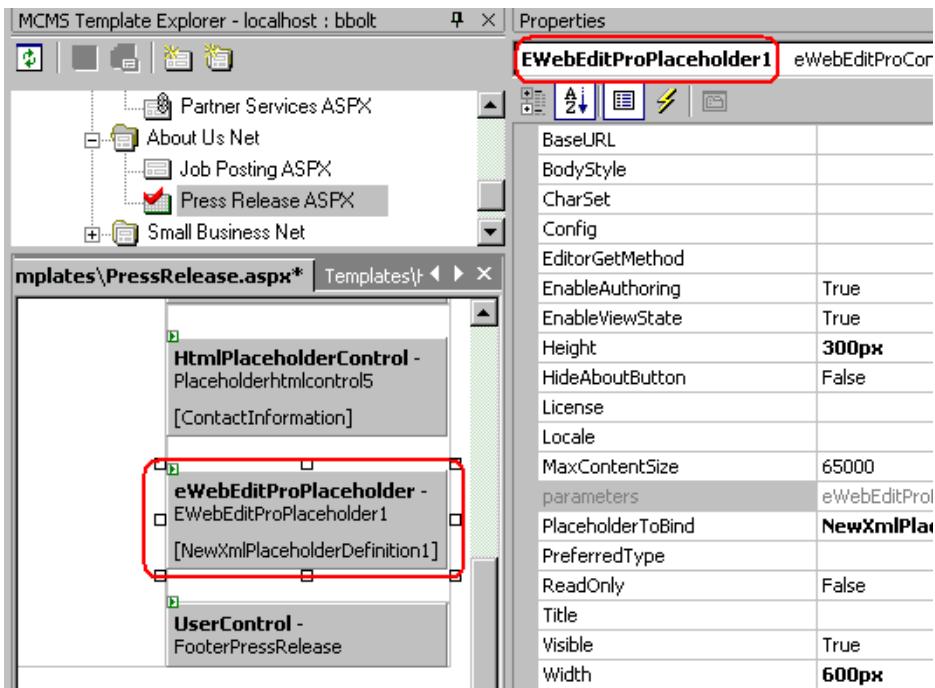
Editing the Properties of the eWebEditPro+XML Placeholder

To modify properties and methods of an instance of the **eWebEditPro+XML** editor, follow these steps.

1. Select the **eWebEditPro+XML** placeholder.
2. Right click the mouse and select **Properties**.



3. The Properties dialog appears.



4. You can edit the following properties. For documentation of the properties, refer to the **eWebEditPro+XML** Developer's Reference Guide (see the chapters "JavaScript Objects" and "Activex Control").

NOTE

Unless stated otherwise, each parameter is of type 'string'. If a parameter is not set, the value assigned in `ewebeditprodefaults.js` is used to create the editor.

- "Property: `BaseUrl`" on page 150
- "Property: `bodyStyle`" on page 159
- "Property: `CharSet`" on page 160
- "Property: `Config`" on page 160
- "Property: `editorGetMethod`" on page 182
- "Property: `hideAboutButton`" on page 162 (boolean)
- "Method: `isChanged`" on page 95
- "Property: `License`" on page 162
- "Property: `Locale`" on page 162
- "Property: `maxContentSize`" on page 168

- "Property: preferredType" on page 169
- "Property: ReadOnly" on page 162 (boolean)
- "Property: Title" on page 163
- "Property: xmllInfo" on page 164
- parameters - see Parameters Object, below

Parameters Object Property

You can use the parameters object property to define

- parameters on the following list that do not appear on the above list
- parameters on the above list that require further definition in the code behind
(for example, MyContent1.parameters.title = "My Title is: " + strTheTitle;)

See Also: "Parameters Object" on page 7

NOTE

Unless stated otherwise, each parameter is of type 'string'. If a parameter is not set, the value assigned in ewebeditprodefaults.js is used to create the editor.

- parameters.baseURL
- parameters.bodyStyle
- parameters.buttonTag.end
- parameters.buttonTag.imageTag.alt
- parameters.buttonTag.imageTag.border (integer)
- parameters.buttonTag.imageTag.height (integer)
- parameters.buttonTag.imageTag.src
- parameters.buttonTag.imageTag.width (integer)
- parameters.buttonTag.start
- parameters.buttonTag.type (note: defaults to "imagelink")
- parameters.buttonTag.tagAttributes
- parameters.buttonTag.value
- parameters.charset
- parameters.clientInstall
- parameters.config
- parameters.editorGetMethod
- parameters.embedAttributes
- parameters.hideAboutButton

- parameters.installPopup.query
- parameters.installPopup.url
- parameters.installPopup.windowFeatures
- parameters.installPopup.windowName
- parameters.license
- parameters.locale
- parameters.maxContentSize (integer)
- parameters.objectAttributes
- parameters.onblur
- parameters.ondblclickelement
- parameters.onexeccommand
- parameters.onfocus
- parameters.popup.query
- parameters.popup.url
- parameters.popup.windowFeatures
- parameters.popup.windowName
- parameters.preferredType
- parameters.readOnly
- parameters.styleSheet (note: defaults to use the stylesheets specified on the page)
- parameters.textareaAttributes
- parameters.title
- parameters.xmlInfo

Declaring the Schema File

eWebEditPro+XML provides a schema file, eWebEditProNet.xsd, that provides proper validation and Intellisense in Visual Studio .NET when in HTML view. The schema only works if it is declared in the ASP.NET page.

To declare it, open the aspx form page, switch to HTML view, and add the xmlns:ewepnet declaration to the body tag as shown.

```
<body xmlns:ewepnet="urn:eWebEditProNet">
```

Integrating eWebEditPro+XML with ColdFusion

Creating Your Own Page

If you want to create a new ColdFusion page and place **eWebEditPro+XML** on that page, the page needs to include these actions.

1. Set up a form.
2. Call the **eWebEditPro+XML** custom tag.
 - Retrieving and Loading Data Designer Content*****
3. Add a submit button.

The rest of this section explains how to complete these tasks.

Note ColdFusion limits the results received from ODBC queries' columns to 64K for performance reasons. It may be possible to edit ColdFusion's settings of your ODBC data source. Refer to your ColdFusion documentation for more information.

Setting Up a Form

When setting up a form, follow these steps.

1. Declare a form.
2. Enter a URL as the action. This defines the page that manipulates the user's input when the user presses the submit button.
3. Enter **Post** as the method.

Here is a sample form declaration.

```
<form action="multiedit.cfm?preview" method="post">
```

Calling the **eWebEditPro+XML** Custom Tag

First Time Installation of **eWebEditPro+XML**

To place the editor on a ColdFusion page, enter a call to the custom tag with the following elements within the form.

```
<CF_ewebeditpro5 name, width, height, initial content>
```

Argument	Description
name	<p>A name for the editor. This is the name of the element that is sent to the server.</p> <p>As of build 2.0.0.30, the CF custom tag includes the attribute <code>EditorName</code> as an alternative to <code>Name</code>. <code>EditorName</code> is needed if the <code>CFMODULE</code> tag is used to instantiate editor instead of <code><CF_ewebeditpro5></code>.</p> <p>Example</p> <pre><cfmodule Name="ewebeditpro5" EditorName="myContent1" Width="95%" Height="220" Value="#initialcontent#"></pre>
width, height	<p>The width and height of the editor in percent or pixels.</p> <ul style="list-style-type: none"> • If a percent, enclose the value in quotes ("") and follow it with a percent sign (%), for example “50%”. • If pixels, quotes are optional, for example, 500.
initial content	<p>If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text.</p> <ul style="list-style-type: none"> • To enter text, enclose it within quotes (""). • To enter a variable, define it elsewhere in the file.

Here is an example of a line that calls the custom tag.

```
<CF_ewebeditpro5 Name= "Editor1" Width="100%" Height="555" Value= "#initial_content#" >
```

In this example, the initial content is defined in the variable `initial_content`, which is defined elsewhere in the file.

You can change ColdFusion custom tag attributes if you want this instance of the editor to be different from the standard. For more information, see [“eWebEditPro+XML’s Custom Tag” on page 758](#).

Adding a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server after entering it. Here is an example of a line that contains a submit button.

```
<input type="submit" name="btnSubmit" value="Submit">
```

NOTE If you create JavaScript to submit the form (instead of the input declaration illustrated above), you must include an `eWebEditPro.save` function prior to the submit function. For example

```
<script>
.
.
eWebEditPro.save( )
myform.submit()
</script>
```

eWebEditPro+XML's Custom Tag

When you install **eWebEditPro+XML**, a ColdFusion custom tag file (`ewebeditpro5.cfm`) is placed in the `CFUSION/Custom Tags` folder on the server. This section describes each attribute in the custom tag.

NOTE If your host does not allow custom tags to be placed in the `CustomTags` file, use the `EditorName` attribute, explained below.

Custom Tag Attributes

The attributes in the custom tag determine many of the key **eWebEditPro+XML** settings, such as maximum content size, editor name and the directory where **eWebEditPro+XML** resides.

Many of these attributes are also stored in **eWebEditPro+XML** files, such as `ewebeditpro.js` and `ewebeditprodefaults.js`. If the same attribute appears in both `ewebeditpro5.cfm` and an **eWebEditPro+XML** file, the value in the `ewebeditpro5.cfm` file takes precedence over that value in the other file.

Attribute	Description
Path	Specifies the path to the directory to which eWebEditPro+XML is installed. By default, this attribute is set to <code>/ewebeditpro5/</code> .
MaxContentSize	The largest number of characters that can be saved in the editor window. If a user enters content that exceeds this size, an error message appears. For more information see “ Property: maxContentSize ” on page 168.

Attribute	Description
Name	<p>The name of the eWebEditPro+XML editor. The name should be a valid JavaScript identifier, so should follow these guidelines.</p> <ul style="list-style-type: none"> • It consists of only ASCII letters and digits, underscores (_) and dollar signs (\$). • The first character cannot be a digit. • Spaces are not permitted. <p>See Also: EditorName attribute</p>
EditorName	<p>An alternative to Name. This is needed if the CFMODULE tag is used to instantiate the editor instead of <CF_ewebeditpro5>.</p> <p>For example:</p> <pre><cfmodule Name="ewebeditpro5" EditorName="myContent1" Width="95%" Height="220" Value="#variables.editor1#"></pre> <p>Why CFMODULE is used to instantiate editor</p> <p>Many hosts do not allow new custom tags to be placed in the ColdFusion CustomTags directory. To work around this problem, place the tag in another directory and call it using <cfmodule template="taglocation/tagname">.</p>
Width	<p>The width of the editor in pixels or a percent. For example, 700 or “100%”.</p>
Height	<p>The height of the editor in pixels or a percent. For example, 400 or “100%”.</p>
Value	<p>If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text.</p> <ul style="list-style-type: none"> • To enter text, enclose it within quotes (""). • To enter a variable, define it elsewhere in the file.

Attribute	Description
License	<p>The license keys of the editor. Separate each with a comma. Ektron provides these keys after purchase. For development purposes, the license keys for 127.0.0.1 and localhost are built into the editor.</p> <p>Note: eWebEditPro+XML displays an Invalid License message if the license key is improperly entered.</p> <p>See Also: The “License Keys” chapter of the eWebEditPro+XML and eWebEditPro+XML+XML installation manual.</p>
Locale	<p>The URL of the localization directory or file, or the locale data itself.</p> <p>For more information, see “Modifying the Language of eWebEditPro+XML” on page 265.</p>
Config	<p>Either the URL of the config XML data or the configuration data itself. Although this ActiveX control property can contain the XML content, it typically refers to an XML file. (For details, see “Managing the Configuration Data” on page 251.)</p>
StyleSheet	<p>Which style sheet file (CSS) to apply to the editor content.</p> <p>For more details, see “Style Sheets” on page 430.</p>
BodyStyle	<p>Set cascading style sheet (CSS) attribute values, such as background color, default font style, size, color and more. The BodyStyle parameter sets any valid CSS style supported by your browser.</p> <p>For more information, see “The Parameters Object” on page 306.</p>
HideAbout Button	<p>Set to “True” to remove the About () button from the toolbar.</p>
WDDX	<p>Available for compatibility with Release 1.8.</p>
onDbClickElement	<p>Double-clicking on a hyperlink, applet, object, image, or table causes this event to fire. Review the <code>ewebeditproevents.js</code> file for an example of how to respond to this event.</p> <p>See Also: “Event: ondblclickelement” on page 186.</p>

Attribute	Description
onExecCommand	<p>The ActiveX control raises the onexeccommand after a toolbar button is pressed, a toolbar drop-down menu item is selected, or a context menu (right-click menu) item is selected.</p> <p><i>See Also:</i> “Event: onexeccommand” on page 186.</p>
onfocus()	<p>An event that fires when the editor gains the focus. onfocus is a standard DHTML event.</p> <p>For details, see “Event: onfocus” on page 187.</p>
onblur()	<p>An event that fires when the editor loses the focus. onblur is a standard DHTML event.</p> <p>For details, see “Event: onblur” on page 187.</p>

Integrating eWebEditPro+XML with PHP

- “Using the Sample Pages” on page 762
- “Creating Your Own Page” on page 762
- “Including a Reference to ewebeditpro.php” on page 762
- “Setting Up a Form” on page 763
- “Placing the Editor on the Form” on page 763
 - “Retrieving and Loading Data Designer Content” on page 765
- “Adding a Submit Button” on page 765

Using the Sample Pages

When you download **eWebEditPro+XML**, Ektron provides sample pages that include the editor. The default location is `www.mywebsite.com/ewebeditpro5/samples/php`.

You should copy these samples to another directory or rename them, and then modify them as needed for your users. If you do not copy them, any changes you make are overwritten when you reinstall or upgrade **eWebEditPro+XML**.

Creating Your Own Page

If you want to create a new PHP page and place **eWebEditPro+XML** on that page, the page needs to include these actions.

1. Include the `ewebeditpro.php` file.
2. Set up a form.
3. Place the editor on the form.
4. Add a submit button.

Finally, you would update your License Key information as needed.

The rest of this section explains how to complete these tasks.

Including a Reference to `ewebeditpro.php`

Your PHP page must contain an `include` command that specifies a relative path to the `ewebeditpro.php` file. Place the `include` line within the page's head tags.

Use this syntax to indicate a relative path to the domain name of this file.

```
<head>
<?php include("../.../ewebeditpro.php"); ?>
</head>
```

Setting Up a Form

When setting up a form, follow these steps.

1. Declare a form.
2. Enter a url as the action. This defines the page that manipulates the user's input when the user presses the submit button.
3. Enter **Post** as the method.

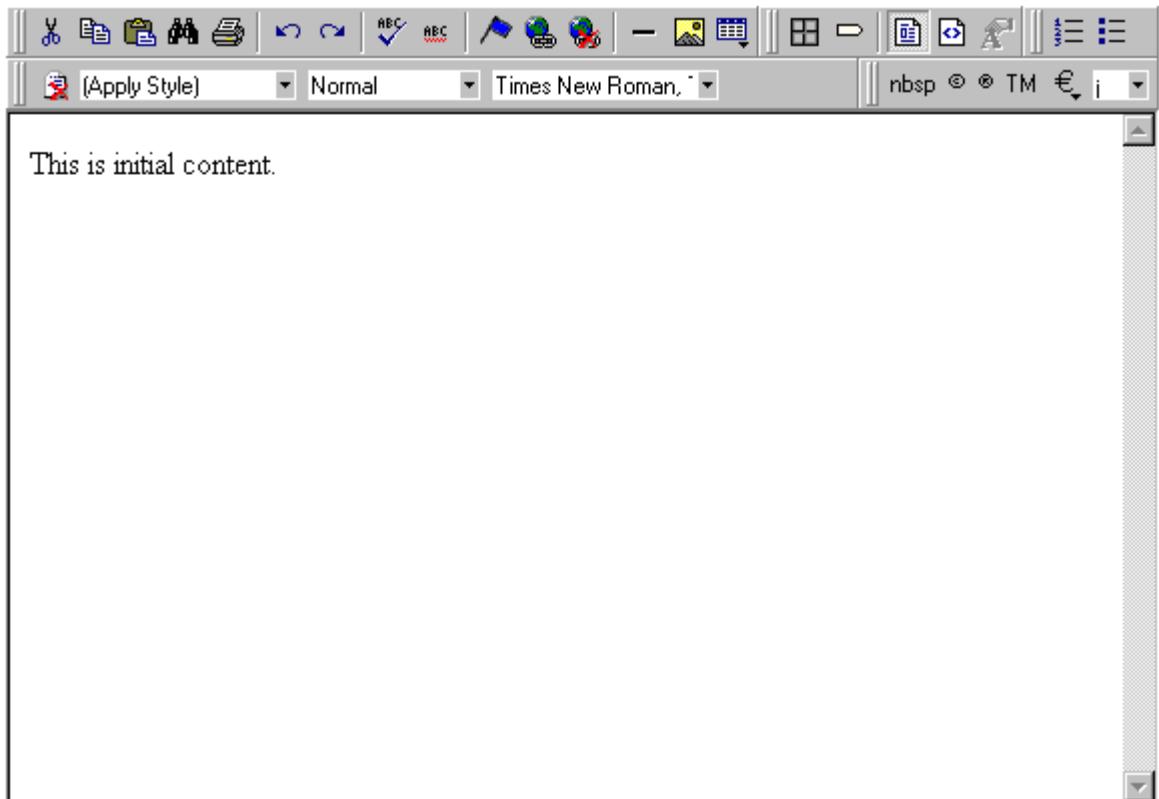
Here is a sample form declaration.

```
<form action="multiedit.php?preview" method="POST">
```

Placing the Editor on the Form

Place the editor within the form tags as a box whose width and height you specify.

This illustration depicts the editor appearing as a box.



For each editor that you want to place on the Web page, you

- change parameter values as needed
- insert the editor

Changing Parameter Values

If you want to change parameters that affect all instances of the editor, edit the ewebeditprodefaults.js file using a standard text editor.

(For a description of each parameter, see “[Customizable JavaScript Files](#)” on [page 291](#).)

To change the parameters only for the instance of **eWebEditPro+XML** that you are placing on the PHP page, enter the following code. In this example, you edit the parameters to remove the About button () from the toolbar.

```
<script language="JavaScript">
eWebEditPro.parameters.reset();
eWebEditPro.parameters.hideAboutButton="False";
</script>
```

If you are placing more than one editor on a page, and you want the parameters for each editor to be different, begin the parameter code with `eWebEditPro.parameters.reset()`. This line restores the parameters to the default values set in ewebeditprodefaults.js.

Inserting the Editor

To place the editor on a PHP page, enter a line with the following elements within the form.

```
<?php echo eWebEditProEditor("field name", height, width, $initial_content); ?>
```

Argument	Description
field name	Enter the name of the field that stores content within quotes (""). It does not matter what the name is, but the field name on the page that retrieves the content must match this name.
height, width	Enter the height and width of the editor in percent or pixels. <ul style="list-style-type: none"> • If a percent, enclose the value in quotes ("") and follow it with a percent sign (%), for example “50%”. • If pixels, quotes are optional, for example, 500.
initial content	If you want some text to appear in the editor the first time a user views it, you can enter text or a variable that contains the text. <ul style="list-style-type: none"> • To enter text, enclose it within quotes (""). • To enter a variable, define it elsewhere in the file.

Here is an example of a line that calls the editor. In this example, the initial content is defined in the variable `strContent1`, which is defined elsewhere in the file.

```
<?php echo eWebEditProEditor("TextHTML1", "100%", 250, $strContent1); ?>
```

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see "Retrieving and Loading Data Designer Content" on page 607 and "PHP" on page 611.

Adding a Submit Button

Add a standard HTML submit button that allows the user to send the content to the Web server after entering it. Here is an example of a line that contains a submit button.

```
<input type="submit" name="btnSubmit" value="Preview">
```

Integrating eWebEditPro+XML Using JavaScript

NOTE

If **eWebEditPro+XML** is used on a platform (that is, a browser or operating system) that does not support **eWebEditPro+XML**, a textarea field automatically appears in its place. No extra work is required to handle unsupported platforms.

Using the Sample Pages

When you download **eWebEditPro+XML**, Ektron provides sample pages that include the editor. The pages are located in the samples folder below the folder to which you installed **eWebEditPro+XML**. The default location in Windows is C:\Inetpub\wwwroot\ewebeditpro5\samples\html.

You should copy these samples to another directory or rename them, and then modify them as needed for your users. If you do not copy them, any changes you make are overwritten when you reinstall or upgrade **eWebEditPro+XML**.

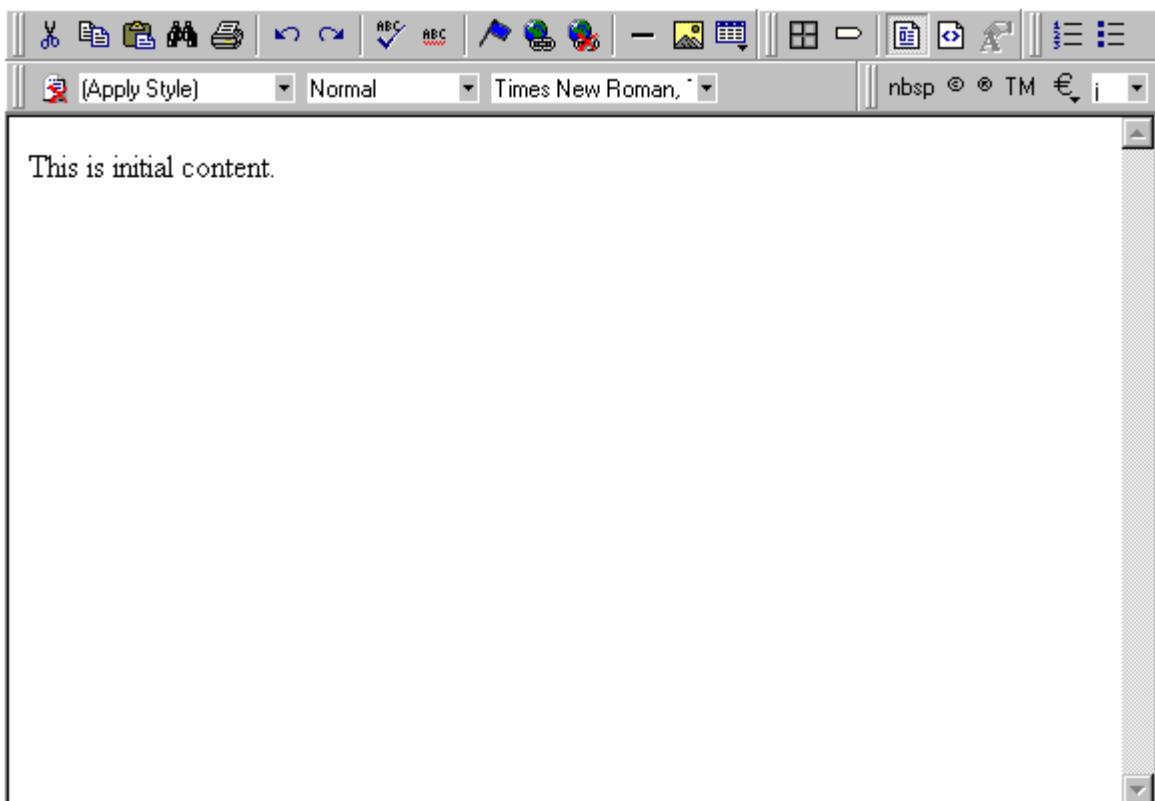
A good sample Web page to study is ewebeditpro5\ewebeditpro.htm.

Formats for Placing the Editor on the Page

You can place the editor as a

- box whose width and height you specify, or
- button that, when pressed, displays a new window with the editor

This illustration depicts the editor appearing as a box.



This illustration depicts the editor as a button. When the user clicks the button, the editor appears in a new window.



Creating Your Own Page

If you want to place **eWebEditPro+XML** on an HTML page, the page needs to include these actions.

1. Create an HTML page with header and body tags.
2. Include the **eWebEditPro+XML** JavaScript file.
3. Set up a form.
4. Modify the parameters (optional).
5. Create an input area.
6. Invoke the editor.

- Retrieving and Loading Data Designer Content

The rest of this section explains how to complete these tasks. Instructions are also provided for [inserting the editor as a button](#) and [encoding characters in the value attribute](#).

Create an HTML Page with Header and Body Tags

Set up a typical HTML page.

```
<HTML>
<head>
<title>eWebEditPro</title>
</head>
<body>
</body>
</HTML>
```

Include the eWebEditPro+XML JavaScript File

Your page must include a reference to the `ewebeditpro.js` file. Place the `src` reference within the head tags.

```
<script language="JavaScript1.2" src="/ewebeditpro5/ewebeditpro.js"></script>
```

Enter a Form Element

Within the body tags, enter a set of form tags. Assign the `method` attribute to the form tags, and `post` as the method's value.

```
<form method="post">
place the editor here
</form>
```

If a form element's method is not set to `post`, an error message appears below the editor.

Changing Parameter Values

If you want to change parameters that affect all instances of the editor, edit the `ewebeditprodefaults.js` file using a standard text editor (see "[The ewebeditprodefaults File](#)" on page 291).

To change the parameters only for the instance of **eWebEditPro+XML** that you are placing on the HTML page, enter the following code.

```
<script language="JavaScript">
eWebEditPro.parameters.parameter= "value";
</script>
```

For example, the following code displays the "about" button on the toolbar.

```
<script language="JavaScript">
eWebEditPro.parameters.hideAboutButton="False";
</script>
```

If you are placing more than one editor on a page, and you want the parameters for each editor to be different, begin the parameter code with `eWebEditPro.parameters.reset()`. This line restores the parameters to the default values set in `ewebeditprodefaults.js`.

Inserting the Editor as a Box

Create a Content Field

Within the form tags, enter a hidden field, text input type tag, or a textarea tag. This example uses a hidden field.

```
<input type=hidden name="MyContent1" value="This is initial content.">
```

See Also: ["Appendix A: Naming the eWebEditPro+XML Editor" on page 778.](#)

NOTE

Please read "Encoding Characters in the Value Attribute" on page 771 for important details about the value attribute.

Declaring a Content Field after Creating the Editor

If the content field (typically a hidden field) appears after the `eWebEditPro.create` code, the following error message appears below the editor.

Content field must be declared prior to creating the editor.

If you must declare the control field after creating the editor, delete the error message before entering the create command.

```
eWebEditProMessages.elementNotFoundMessage= " ";
eWebEditPro.create(...);
```

The content will still be loaded, but it cannot check to see if the content field exists.

Creating the Editor

Within the input area, place the editor using the following JavaScript.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.create("field name", width, height);
//-->
</script>
```

Argument	Description
field name	Enter the name of the field within quotes (" "). The field name must match the value of the <code>name</code> attribute in the input type tag.
width, height	Enter the width and height of the editor in percent or pixels. If a percent, enclose the value in quotes (" ") and follow it with a percent sign (%), for example "50%". If pixels, quotes are optional, for example, 500.

For example

```
<script language="JavaScript1.2">
<!--
eWebEditPro.create("MyContent1", 700,150);
//-->
</script>
```

The content is automatically loaded into the editor and automatically saved when the form is submitted.

NOTE If you submit by calling the `form.submit` method, you must manually save the content by calling `eWebEditPro.save()` just prior to calling `form.submit`. To learn how content is loaded and saved, see "Loading the Content" on page 773.

To access the ActiveX control via JavaScript, once an instance of the editor is created, use the **eWebEditPro+XML** JavaScript object.

`eWebEditPro.theeditorname`

The ActiveX control should only be accessed after the **eWebEditPro+XML** `onready` event fires. For example

```
eWebEditPro.instances.MyContent1.editor.pasteHTML("<HR>"); //  
insert horz rule
```

The `pasteHTML` method inserts HTML content into the editor. For more information on the `pasteHTML` and other methods, properties and events of the **eWebEditPro+XML** ActiveX control, see "ActiveX Control" on page 309.

Retrieving and Loading Data Designer Content

If you want to retrieve and load data designer content into the editor, see "Retrieving and Loading Data Designer Content" on page 607 and "JavaScript (eWebEditPro Object)" on page 611.

Inserting the Editor as a Button

Entering a Field

Within the form tags, enter a hidden field or text input type tag, or a or textarea tag. This example uses a textarea declaration.

```
<textarea name="tal" cols=80 rows=5>Click 'Edit' to edit this content with the eWebEditPro editor.
</textarea>
```

NOTE If you decide to use hidden field or a text field, read "Encoding Characters in the Value Attribute" on page 771 for important details about the value attribute.

Entering the Button

To add the button to the page, enter a line with the following elements.

```
<script language="JavaScript1.2">
<!--
eWebEditPro.createButton("button name", "textarea name");
//-->
</script>
```

Argument	Description
button name	Enter a button name within quotes (" "). This is the field name of the button that by default is <input type=button...>. To change the button type, see "Customizing the Popup Button" on page 253.
textarea name	Enter the name of the textarea within quotes (" ").The field name must match the field named in the textarea declaration.

For example

```
<script language="JavaScript1.2">
<!--
eWebEditPro.createButton("btnEdit", "tal");
//-->
</script>
```

NOTE To edit the button text, open the ewebeditpromessages.js file using a standard text editor. Within that file, edit the text within quotes that follows **popupButtonCaption:**

Encoding Characters in the Value Attribute

Initial content for the editor is typically stored in the value attribute of a hidden field. For example,

```
<input type="hidden" name="MyContent1" value="This is the initial content.">
```

If the content includes a quote ("), greater or less than character (<>), or an ampersand (&), the browser prematurely terminates the display of the content. For example, the input declaration

```
<input type="hidden" name="MyContent1" value="Characters that need to be encoded: " & <tag">
```

would display the following text in the editor

Characters that need to be encoded:

This problem occurs because the browser cannot distinguish between one of these characters and the delimiters of the value attribute.

Also, if you use single quotes to delimit the value attribute, which is not recommended, you need to encode all single-quote characters.

To solve this problem, you must encode these characters when storing them in a hidden field. You would insert the character's *entity* or *character reference* in place of the actual character in the value field.

The following table lists the characters and corresponding entity and character reference values.

Character	Entity	Character Reference	Comments
&	&	&	Must be encoded first.
>	<	<	
<	>	>	
"	"	"	Value attribute must be quoted with ", not '.

NOTE The order in which characters are encoded is important. The ampersand (&) must be encoded before you encode the other characters.

How the Server Converts Characters

Your Web application server must convert these characters. For example, ASP offers the Server.HTMLEncode function. If your environment does not provide such a function, you need to write it. It is straightforward and requires the use of a string substitution function. The pseudo code to encode these characters appears below.

```
String strContent
strContent = ReplaceString(strContent, "&", "&amp;")
strContent = ReplaceString(strContent, "<", "&lt;")
strContent = ReplaceString(strContent, ">", "&gt;")
strContent = ReplaceString(strContent, "\"", "&quot;")
```

Encoding the Single Quote

We recommend surrounding the value attribute with double quotes, but if you decide to use single quotes, you must encode the single quote character (also known as an apostrophe).

Character	Entity	Character Reference	Comments
'	' (but see comments)	'	&apos; is for XML parsers but may not be supported by an HTML browser. Therefore, the character reference (') is preferred, because HTML browsers <i>and</i> XML parsers support it.

Content Stored in a Textarea Field

When stored in a TEXTAREA field, the greater/less than characters (<>) do not need to be encoded, because TEXTAREA does not use a value attribute. The double-quote ("), single-quote (') and ampersand (&) characters should be encoded in a TEXTAREA field, although most browsers will accept them without encoding.

Loading the Content

Content is loaded into the editor during the page's onload event, which invokes the eWebEditPro.load method. The method copies content from the hidden field (or other HTML element) to the editor.

To prevent loading, set window.eWebEditProLoadHandled to **true**.

```
window.eWebEditProLoadHandled=true;
```

Detecting the Load Method

To detect when the load method is being invoked, two **eWebEditPro+XML** events are fired at this time.

- onbeforeload
- onload

If the onbeforeload event handler returns false, it terminates the load method and the onload event.

For example,

```
eWebEditPro.onbeforeload="return confirm('Do you want to load?');"
eWebEditPro.onload="alert('Done loading.');
eWebEditPro.create(...);
```

NOTE

In Netscape, the alert may not function during onload events.

Manually Loading Content into the Editor

To let the user manually load content, use this syntax.

```
window.eWebEditProLoadHandled=true
eWebEditPro.create(...)
</script>
<input type=button value="load" onclick="eWebEditPro.load()">
```

The `eWebEditPro.load` method loads all instances of the editor on the page. To load just one, you can use `eWebEditPro.instances[n].load()`. Or, you can pass the content using `eWebEditPro.instances[n].load(strContent)`.

If you use `eWebEditPro.instances[n].load()`, the `n` within square brackets is either the name of the editor used when creating it (for example, `eWebEditPro.create("EditorName", ...)`) or an index number (0, 1, 2, etc., where 0 is the first editor created).

See Also: “[Appendix A: Naming the eWebEditPro+XML Editor](#)” on page 778.

For example

```
eWebEditPro.create("Summary", 700, 200);
eWebEditPro.create("Teaser", 700, 300);
eWebEditPro.create("Desc", 700, 400);
...
eWebEditPro.instances["Desc"].load();
```

or

```
eWebEditPro.instances[2].load();
```

Saving the Content

Content is saved (that is, copied to the hidden field) during the form’s `onsubmit` event, which invokes the `eWebEditPro.save` method. This method copies the content from the editor to the hidden field (or other HTML element). In Internet Explorer, the content is also saved when the page is unloaded.

NOTE	The <code>eWebEditPro.save</code> method saves content to a temporary cache in the browser. The content is saved permanently when the form is submitted and its fields are posted to the server.
-------------	--

To prevent saving, set the form’s `eWebEditProSubmitHandled` method to **true**.

```
document.yourformname.eWebEditProSubmitHandled=true;
```

Detecting when the Save Method is Invoked

To detect that the save method is being invoked, two **eWebEditPro+XML** events are fired at this time.

- `onbeforesave`
- `onsave`

Terminating the Save Method

Returning false in the onbeforesave event handler terminates the save method and the onsave event.

For example

```
eWebEditPro.onbeforesave="return confirm('Do you want to save?');"
eWebEditPro.onsave="alert('Done saving');
```

Saving Content Manually

To manually save content, use this code.

```
function mysubmit(){
    eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE;
    eWebEditPro.save();
    document.myform.submit();
```

Closing a Window without Saving Content

To close a window (that is, cancel) without saving the content to the hidden field, use this code.

```
eWebEditPro.actionOnUnload = EWEP_ONUNLOAD_NOSAVE;
self.close();
```

See Also: "Preventing the Save Caused by an onbeforeunload Event" on page 775

Prevent Detecting the onsubmit Event

To prevent automatic saving of the content to the hidden field when a submit button is pressed, use this code. It must appear *prior* to creating the editor on the page.

```
document.myform.eWebEditProSubmitHandled=true;
eWebEditPro.create(...);
```

See Also: "Preventing the Save Caused by an onbeforeunload Event" on page 775

Prevent Detecting the onbeforeunload/onunload Event

To prevent automatically saving the content to the hidden field when the Web page is unloaded, use this code. It must appear *prior* to creating the editor on the page.

```
window.eWebEditProUnloadHandled=true;
eWebEditPro.create(...);
```

See Also: "Preventing the Save Caused by an onbeforeunload Event" on page 775

Preventing the Save Caused by an onbeforeunload Event

Sometimes, a user performs an action that causes the current window to close. For example, he clicks the small X in the top right corner.

When such an action occurs, Internet Explorer fires an `onbeforeunload` event, which saves the content to the hidden field. If you want to intercept the event and let the user decide whether or not to save the content at that time, use the following code. Note that the `onbeforesave` method is inserted prior to the page create event.

```
eWebEditPro.onbeforesave = MySaveCheck;
eWebEditPro.create("myeditor", "100%", "90%");
```

Then, insert the following function on the page to detect whether to allow the editor to save. You do *not* want to never save because that would mean that new content is never saved.

```
// Return false to abort the save.
function MySaveCheck()
{
    if(ConditionsAllowSave())
        return(true); // true allow the save to continue
    else
        return(false); // false stops the save
}
```

Finally, if you want suppress the warning message about the save, use the following code.

```
eWebEditProMessages.confirmAway = null;
```

Saving from One Instance of the Editor

The `eWebEditPro.save` method saves all instances of the editor on the page. To save just one, use `eWebEditPro.instances[n].save()`. The `n` within square brackets is either the name of the editor used when creating it (for example, `eWebEditPro.create("EditorName", ...)`) or an index number (0, 1, 2, etc., where 0 is the first editor created).

See Also: “Appendix A: Naming the eWebEditPro+XML Editor” on page 778.

For example

```
eWebEditPro.create("Summary", 700, 200);
eWebEditPro.create("Teaser", 700, 300);
eWebEditPro.create("Desc", 700, 400);
...
eWebEditPro.instances["Desc"].save();
```

or

```
eWebEditPro.instances[2].save();
```

Alternatively, you can retrieve content by passing an object to the `save` method. To do this, set the object's `value` property to receive the content.

For example

```
var objContent = new Object();
objContent.value="";
eWebEditPro.save(objContent);
```

`.objContent.value` now stores the content.

Detecting When the Popup Editor is Activated

Similarly, when using a popup editor with **eWebEditPro+XML**.createButton(), there are two **eWebEditPro+XML** events that fire when the button is pressed and when the popup window is closed.

- onbeforeedit
- onedit

For example

```
eWebEditPro.onbeforeedit="return confirmed('Do you want to edit?');"  
eWebEditPro.onedit="alert('Done editing')";  
eWebEditPro.createButton(...);
```

Testing the Page

After you create your Web page, test it to verify that it works as planned. When testing the page, you cannot simply double click the .html file. Instead, you must type the following url into the Web browser's address field:

`http://localhost/ewebeditpro5/folder/filename.htm`

For example, if the file is named mytest.htm and it is located in a folder named ewebeditpro, enter this url into your browser:

`http://localhost/ewebeditpro5/mytest.htm`

Appendices

Appendix A: Naming the eWebEditPro+XML Editor

When you are naming the **eWebEditPro+XML** editor, the name must be a valid JavaScript identifier. As a result, the name must follow these guidelines.

- It consists of only ASCII letters and digits, underscores (_) and dollar signs (\$).
- The first character cannot be a digit.
- Spaces are not permitted.
- Do *not* assign **BASE** as the name. BASE conflicts with JavaScript in the editor, so you should avoid this name.

Appendix B: Error Messages

Error Message	Cause	How to Resolve	Audience	Where Message is Defined
ActiveBar 2.0 32-Bit ActiveX Thank you for choosing to evaluate ActiveBar from Data Dynamics Ltd. This version of the software is for Evaluation Purposes Only and may be used for up to 30 days to determine if it meets your requirements...	This error message displays if Internet Explorer cannot access the ewebeditpro.lpk file when the editor appears in a Web page. Internet Explorer looks to the ewebeditpro.lpk file for ActiveBar license information. An LPK file is the standard mechanism to license ActiveX controls for use with Internet Explorer. ActiveBar is an ActiveX® control used by eWebEditPro+XML .	Remove the Evaluation message by clicking the OK button. eWebEditPro+XML will continue to function normally even after 30 days. To suppress the Evaluation message, try the following ideas. For Internet Explorer, ensure that <ul style="list-style-type: none"> • the ewebeditpro.lpk file is not corrupt and accessible. It must reside in the server directory where eWebEditPro+XML was installed (for example, /ewebeditpro5/). • a firewall does not block it. • the server permits .lpk file extensions. One way to do this is to type the URL to the ewebeditpro.lpk in the browser's address bar. • the server does not append information to the end of the LPK file. Running the client installation program should suppress the evaluation message. Since the client installation program is required for Netscape, the Evaluation message should not appear when using Netscape.	End user	by ActiveBar
A license is required for host:	The license key does not match the host name.	The developer must specify a valid license key.	End user/Developer	locale0000b.xml

Error Message	Cause	How to Resolve	Audience	Where Message is Defined
Click OK to preserve changes when moving to another page. Click Cancel to discard changes.	Unloading a Web page with the editor prompts to cache the content in the content field. (Only with IE.)	The end user can click OK or Cancel. The developer can change the value of actionOnUnload.	End user	ewebeditpromessages.js
Content is too large to save. Please reduce the size and try again.	The size of the HTML content in the editor is larger than the amount specified in maxContentSize.	The end user can reduce the content. The developer has several options. For more information, see http://www.ektron.com///support/ewebeditprokb.cfm?doc_id=1326 and http://www.ektron.com///support/ewebeditprokb.cfm?doc_id=1204	End user	ewebeditpromessages.js
Error uploading the selected file.
The uploading of files may not be allowed at this location. Please verify that the connection settings and server permissions are correct.	The server is not allowing the uploading of files. The wrong server may have been specified in the login information. The login account may not have upload permissions.	<ol style="list-style-type: none"> Verify that the server is the correct server used for uploading files. Contact the site administrator to ensure that the login account has upload permissions. 	End user	locale0000b.xml
eWebEditPro+XML cannot clean the document until these errors are fixed.	The HTML content is corrupt and could not be adequately cleaned.	Manually fix the corruption and try again.	End user	locale0000b.xml
eWebEditPro+XML is not installed. Click to install eWebEditPro+XML.	The editor has not been installed yet.	Install the editor using the client installation program.	End user	ewebeditpromessages.js

Error Message	Cause	How to Resolve	Audience	Where Message is Defined
Internet Explorer 4.0 or later is required.	IE 3.x or older is being used.	IE 4.01 or later is required.	End user	locale0000b.xml
Invalid License	<p>The license key is invalid. Likely reasons include</p> <ol style="list-style-type: none"> 1. The domain name in the URL does not match one of the license keys 2. The license key is expired 3. No license key is specified 4. The license key is for another product or version 5. The license key is corrupt 	<ol style="list-style-type: none"> 1. The domain name in the browser's address or location bar must match one of the license keys. For example, http://www.ektron.com matches www.ektron.com?123456, but http://www.ektron.com does not match 123.045.067.089?123456. Use either the name specified in the license key or purchase another license key for the domain name. 2. Purchase a license key. 3. Specify a valid license key. See <i>Also:</i> the Knowledge Base article "Error Message: Invalid license with no license keys in box" at http://www.ektron.com//support/ewebeditprokb.cfm?doc_id=936. 4. Purchase a license key for this product or version. eWebEditPro+XML 2.0 license keys are not valid for eWebEditPro+XML 3, etc. 5. Specify the entire license key and ensure all the numbers are correct. For example, www.ektron.com?123456-20, not just 123456-20. 	<p>End user. Appears in the About box, which pops up if a valid license is not specified.</p>	locale0000b.xml
License is expired for date:	The license key is expired.	The developer must specify a valid license key.	End user/Developer	localev20000.xml

Error Message	Cause	How to Resolve	Audience	Where Message is Defined
Sorry, the connection could not be established. Please verify that the login and connection information are correct.	The connection to the server could not be established. This could be caused by incorrect server address or login information.	Verify that the server domain, login name and password are correct. If they are, contact the administrator of the remote site to verify that the login information is correct.	End user	localev20000.xml
The editor was not able to create the DHTML Editor. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user	localev20000.xml
The editor was not able to create the HTML Source View Editor. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user	localev20000.xml
The editor was not able to create the Toolbar. Please run the client installation or contact your system administrator.	The editor was not able to create a critical component because one or more required files are missing or corrupt.	Try installing using the client installation program.	End user	localev20000.xml
The form method must be set to " post ". For example, <form method="post">. The submit will fail using "get".	The form's method is not set to post.	The developer must set the method to post.	Developer	ewebeditpromessages.js

Error Message	Cause	How to Resolve	Audience	Where Message is Defined
The page content is still initializing. Please wait...	The Web page with the editor is still loading and initializing when the end user pressed a toolbar button. This message is typically only seen when using Netscape.	Wait a few seconds and try again.	End user	localev2000.xml
The selected file is too large to allow an upload. The maximum size is	The size of the target upload file exceeds the upload limits defined by the site administrator.	<ol style="list-style-type: none"> Select a file that is less than the maximum allowed size. Ask the site administrator to increase the size limit specified in the configuration data. 	End user	localev2000.xml
There is excessive HTML code that may prevent you from changing text format.	Prompt to clean Office/Word 2000 content	It is recommended that you clean the content.	End user	localev2000.xml
There was an error in the dialog. The client installation for the editor may need to be run to correct the issue.	The editor was not able to open a dialog window because one of the required files is missing or out of date.	Try installing using the client installation program.	End user	localev2000.xml
Unable to check spelling. Microsoft Word 97 or later is required.	Spell checking is not supported because Word 97 or later is not installed or cannot be accessed.	Install Microsoft Word 97 or later.	End user	localev2000.xml

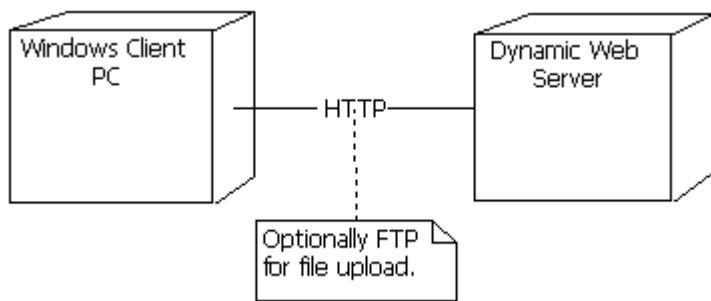
Error Message	Cause	How to Resolve	Audience	Where Message is Defined
Unable to find content field (typically a hidden field) within a form. Please check the following: <ul style="list-style-type: none">• Form tag is required• Content field is required and must match the name specified when creating the editor• Content field must be declared prior to creating the editorName specified:	The editor could not find the content field.	Please check the following: <ul style="list-style-type: none">• Form tag is required• Content field is required and must match the name specified when creating the editor• Content field must be declared prior to creating the editor	Developer	ewebeditpromessages.js
Unable to run in container:	The editor is being used in an application other than a Web browser.	The editor can only be used in a browser.	Developer	localev20000.xml
Unable to save. Continue and lose content?	The editor is unable to save the content in the content field, typically because the window with the content field was closed.	The end user can copy the content to the clipboard to preserve it.	End user	ewebeditpromessages.js

Appendix C: eWebEditPro+XML Architecture

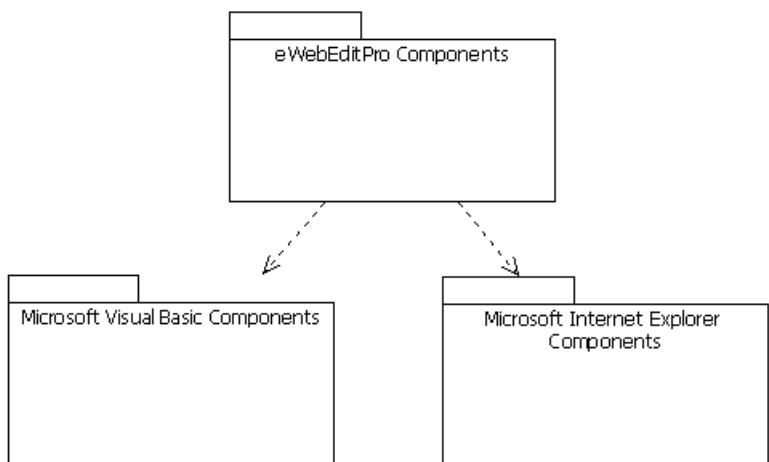
The **eWebEditPro+XML** editor is a browser plug-in. It runs in a Web browser on the client computer.

The editor's content is stored in the client browser. As a result, the editor does not need to make a direct HTTP connection to the server to manage the content. The content is transferred by the browser itself using standard form elements.

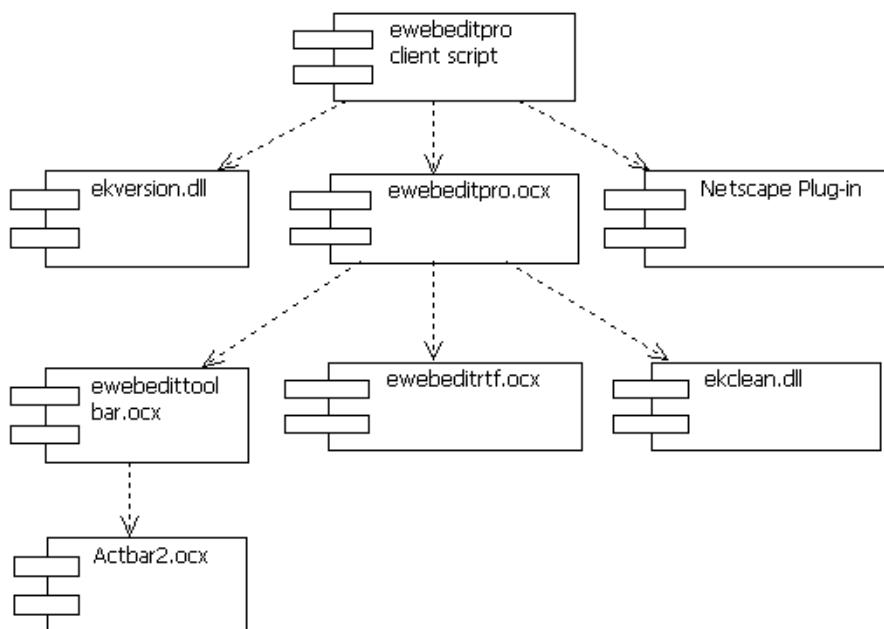
The editor typically retrieves configuration and localization information by downloading files from the server using HTTP or HTTPS. Optionally, the editor can be configured to upload files (for example, images) to the server using FTP. Images may also be uploaded using HTTP in a standard Web form. The following illustration indicates these relationships.



The **eWebEditPro+XML** editor uses standard Microsoft Visual Basic components and standard Microsoft Internet Explorer components. The IE components are used to edit the content even when the editor is in a page loaded by Netscape.



The editor is placed in a Web page using a native dynamic language (for example, ASP, JSP) or JavaScript. The dynamic languages are just a thin wrapper around JavaScript. The JavaScript then creates an instance of the **eWebEditPro+XML** editor in the browser. The following diagram shows the relationship between the client-side script and the other OCX and DLL files that make up the **eWebEditPro+XML** editor.



Appendix D: Automatic Upload File Types

This appendix lists file types and their corresponding numeric values, which are used in Automatic Upload feature. For more information, see "Automatic Upload Object" on page 562.

When a file is automatically uploaded, **eWebEditPro+XML** normally supplies a numeric code to identify the file type. The code lets a server script determine the type of file being uploaded, so that it can determine how to organize and store the file.

The file type name follows the HTML convention, where a GIF file is an "image/gif" type.

NOTE

The numeric code -1 is assigned to an unknown file type.

The following tables list all file types and their corresponding numeric codes, organized into the following categories.

- "Images" on page 788
- "Audio" on page 790
- "Video" on page 791
- "Text" on page 792
- "Application (file for a specific application)" on page 793
- "Other" on page 800

Images

File extension	MIME file type	Numeric value
jpg	image/jpeg	0
gif	image/gif	1
png	image/png	2
jpeg	image/jpeg	3
tif	image/tiff	4
bmp	image/x-ms-bmp	5
tga	image/x-targa	6
emf	image/x-emf	7
wmf	image/x-wmf	8
img	image/x-img	9
pic	image/x-pict	10
pcx	image/x-pcx	11
jpe	image/jpeg	12
tiff	image/tiff	13
cgm	image/cgm	14
cmx	image/x-cmx	15
dsf	image/x-mgx-dsf	16
dwg	image/x-dwg	17
dxf	image/x-dxf	18
fif	image/fif	19

File extension	MIME file type	Numeric value
g3f	image/g3fax	20
ief	image/ief	21
mil	image/x-cals	22
pbm	image/x-portable-bitmap	23
pcd	image/x-photo-cd	24
pgm	image/x-portable-graymap	25
pict	image/x-pict	26
pnm	image/x-portable-anymap	27
ppm	image/x-portable-pixmap	28
ras	image/cmu-raster	29
ras	image/x-cmu-raster	30
rgb	image/x-rgb	31
svf	image/vnd.svf	32
wi	image/wavelet	33
xbm	image/x-xbitmap	34
xpm	image/x-xpixmap	35
xwd	image/x-xwindowdump	36

Audio

File extension	MIME file type	Numeric value
abs	audio/x-mpeg	100
aif	audio/x-aiff	101
aifc	audio/x-aiff	102
aiff	audio/x-aiff	103
au	audio/basic	104
es	audio/echospeech	105
kar	audio/midi	106
mid	audio/midi	107
midi	audio/midi	108
mp2	audio/mpeg	109
mp2a	audio/x-mpeg-2	110
mp3	audio/mpeg	111
mpa	audio/x-mpeg	112
mpa2	audio/x-mpeg-2	113
mpega	audio/x-mpeg	114
mpga	audio/mpeg	115
ra	audio/x-realaudio	116
ram	audio/x-pn-realaudio	117
rm	audio/x-pn-realaudio	118
rpm	audio/x-pn-realaudio-plugin	119

File extension	MIME file type	Numeric value
snd	audio/basic	120
tsi	audio/TSP-audio	121
vox	audio/voxware	122
wav	audio/x-wav	123

Video

File extension	MIME file type	Numeric value
avi	video/x-msvideo	200
fli	video/x-fli	201
mov	video/quicktime	202
movie	video/x-sgi-movie	203
mp2v	video/mpeg-2	204
mpe	video/mpeg	205
mpeg	video/mpeg	206
mpg	video/mpeg	207
mpv2	video/mpeg-2	208
qt	video/quicktime	209
vdo	video/vdo	210
viv	video/vivo	211
vivo	video/vnd.vivo	212

Text

File extension	MIME file type	Numeric value
asc	text/plain	300
c	text/plain	301
cc	text/plain	302
css	text/css	303
etx	text/x-setext	304
f	text/plain	305
f90	text/plain	306
h	text/plain	307
hh	text/plain	308
htm	text/html	309
html	text/html	310
js	text/javascript	311
ls	text/javascript	312
m	text/plain	313
mocha	text/javascript	314
rtf	text/rtf	315
rtx	text/richtext	316
sgm	text/sgml	317
sgml	text/sgml	318
talk	text/x-speech	319

File extension	MIME file type	Numeric value
tsv	text/tab-separated-values	320
txt	text/plain	321
vbs	text/vbscript	322
xml	text/xml	323

Application (file for a specific application)

File extension	MIME file type	Numeric value
ai	application/postscript	400
ano	application/x-annotator	401
asn	application/astound	402
asp	application/x-asap	403
axs	application/x-olescript	404
bcpio	application/x-bcpio	405
bin	application/octet-stream	406
ccad	application/clariscad	407
ccv	application/ccv	408
cdf	application/x-netcdf	409
class	application/octet-stream	410
cpio	application/x-cpio	411
cpt	application/mac-compactpro	412

File extension	MIME file type	Numeric value
csh	application/csh	413
css	application/x-pointplus	414
db	application/octet-stream	415
dcr	application/x-director	416
dir	application/x-director	417
dms	application/octet-stream	418
doc	application/msword	419
doc	application/x-frameset	420
drw	application/drafting	421
dvi	application/x-dvi	422
dxr	application/x-director	423
eps	application/postscript	424
evy	application/envoy	425
exe	application/octet-stream	426
ez	application/andrew-inset	427
faxmgr	application/x-fax-manager	428
faxmgrjob	application/x-fax-manager-job	429
fm	application/x-frameset	430
frame	application/x-frameset	431
frm	application/x-frameset	432
gtar	application/x-gtar	433

File extension	MIME file type	Numeric value
gz	application/x-gzip	434
hdf	application/x-hdf	435
hqx	application/mac-binhex40	436
icnbk	application/x-iconbook	437
igs	application/iges	438
ins	application/x-net-install	439
ins	application/x-insight	440
insight	application/x-insight	441
inst	application/x-install	442
ips	application/x-ipscript	443
ipx	application/x-ipix	444
latex	application/x-latex	445
lcc	application/fastman	446
lha	application/octet-stream	447
lic	application/x-enterlicense	448
lsp	application/x-lisp	449
lzh	application/octet-stream	450
ma	application/mathematica	451
mail	application/x-mailfolder	452
man	application/x-troff-man	453
mbd	application/mbedlet	454

File extension	MIME file type	Numeric value
me	application/x-troff-me	455
mif	application/x-mif	456
mpp	application/vnd.ms-project	457
ms	application/x-troff-ms	458
nc	application/x-netcdf	459
niff	application/vnd.music-niff	460
oda	application/oda	461
ods	application/x-oleobject	462
p3d	application/x-p3d	463
pac	application/x-ns-proxy-autoconfig	464
pcn	application/x-pcn	465
pdf	application/pdf	466
pgn	application/x-chess-pgn	467
pl	application/x-perl	468
pot	application/mspowerpoint	469
pp	application/x-ppages	470
ppages	application/x-ppages	471
pps	application/mspowerpoint	472
ppt	application/mspowerpoint	473
ppz	application/mspowerpoint	474
pre	application/x-freelance	475

File extension	MIME file type	Numeric value
prt	application/pro_eng	476
ps	application/postscript	477
rad	application/x-rad-powermedia	478
roff	application/x-troff	479
sc	application/x-showcase	480
scm	application/x-lotusscreencam	481
sea	application/x-stuffit	482
set	application/set	483
sgi-lpr	application/x-sgi-lpr	484
sh	application/sh	485
shar	application/shar	486
sho	application/x-showcase	487
show	application/x-showcase	488
showcase	application/x-showcase	489
sit	application/x-stuffit	490
skd	application/x-koan	491
skm	application/x-koan	492
skp	application/x-koan	493
skt	application/x-koan	494
slate	application/slate	495

File extension	MIME file type	Numeric value
slides	application/x-showcase	496
smgl	application/sgml	497
smi	application/smil	498
smil	application/smil	499
sol	application/solids	500
spl	application/futuresplash	501
src	application/x-wais-source	502
step	application/STEP	503
stl	application/SLA	504
stp	application/STEP	505
sv4crc	application/x-sv4crc	506
svd	application/vnd.svd	507
swf	application/x-shockwave-flash	508
t	application/x-troff	509
tar	application/x-tar	510
tardist	application/x-tardist	511
tcl	application/tcl	512
tex	application/x-tex	513
texi	application/x-texinfo	514
texinfo	application/x-texinfo	515
tr	application/x-troff	516

File extension	MIME file type	Numeric value
tsp	application/dsptype	517
unv	application/i-deas	518
ustar	application/x-ustar	519
uu	application/octet-stream	520
v4cpio	application/x-sv4cpio	521
v5d	application/vis5d	522
vcd	application/x-cdlink	523
vda	application/vda	524
wb	application/x-inpview	525
wba	application/x-webbasic	526
wkz	application/x-wingz	527
wpd	application/wordperfect5.1	528
wsrc	application/x-wais-source	529
xlc	application/vnd.ms-excel	530
xll	application/vnd.ms-excel	531
xlm	application/vnd.ms-excel	532
xls	application/vnd.ms-excel	533
xlw	application/vnd.ms-excel	534
xsl	application/ms-excel	535
zip	application/zip	536
ztardist	application/x-ztardist	537

Other

File extension	MIME file type	Numeric value
3dmf	x-world/x-3dmf	1000
dwf	drawing/x-dwf	1001
ice	x-conference/x-cooltalk	1002
iges	model/iges	1003
iv	graphics/x-inventor	1004
mesh	model/mesh	1005
mime	www/mime	1006
mmid	x-music/x-midi	1007
msh	model/mesh	1008
opp	x-form/x-openscape	1009
pdb	chemical/x-pdb	1010
silo	model/mesh	1011
svr	x-world/x-svr	1012
vrml	model/vrml	1013
vrw	x-world/x-vream	1014
vts	workbook/formulaone	1015
wfx	x-script/x-wfxclient	1016
wrl	model/vrml	1017
wvr	x-world/x-wvr	1018
xyz	chemical/x-pdb	1019

Index

Symbols

<P> tags
removing 356

A

about window, displaying 200
absolute positioning an image or table 200
accessibility
 dialogs that allow input 731
 making content comply with standards 727
actionOnUnload, JavaScript object property 177
Active Tag method 50
ActiveBar 779
ActiveSchema property, XML interface object 136
ActiveTag method, XML interface object 50
ActiveX
 control properties
 BaseUrl 166
 CharSet 160
 Config 160
 Disabled 160
 Get WDDX() As String 162
 hideAboutButton 162
 IsDirty 162
 License 162
 Locale 162
 ReadOnly 162
 srcPath 163
 StyleSheet 163
 Title 163
 versionInstalled 163
 xmlInfo 164
events
 onblur 187
 ondblclickelement 186
 onexeccommand 186
 onfocus 187
methods 310
 disableAllStyleSheets 69
 ExecCommand 81
 Focus 82
 getBodyHTML 83
 getBodyText 84
 getDocument 85
 getHeadHTML 88
 getProperty 90
 getPropertyBoolean 90
 getPropertyInteger 91

 getPropertyString 91
 getSelectedHTML 92
 getSelectedText 92
 isEditorReady() As Boolean 97
 MediaFile() As Media File Object 106
 pasteHTML 109
 pasteText 110
 setBodyHTML 119
 setDocument 120
 setHeadHTML 123
 setProperty 124
 style sheets 52, 53, 104, 115
 addInlineStyle 51
 BodyStyle 61
 ClearStylesFromTags 63
 disableStyleSheet 69
 GetActiveStyleSheetTitles 83
 PopulateTagsWithStyles 110
 ShowActiveStylesDetails 126
 Toolbars() As Toolbar Control Object 130
 XMLProcessor() As XML Object 133
addEventHandler, JavaScript object method 50
addInlineStyle, ActiveX style sheet method 51
addLinkedStyle Sheet, ActiveX style sheet method 52, 53, 104, 115
AddSchema method, XML interface object 54
AllowEdit property, XML Data object 142
AllowSubDirectories, image selection object property 149
allowUpload
 Automatic Upload Object 148
AllowUpload attribute, autoupload element
 autoupload element, AllowUpload attribute 501
AllowUpload, image selection object property 150
AnySchemasLoaded method, XML interface object 55
ApplyTag method, XML interface object 56
 architecture of eWebEditPro 785
array, instanceTypes 305
ashow attribute
 XML feature 686
AskOpenFile method, WebIm ageFX 57
AskSaveAs method, WebIm ageFX 57
AskSelectColor method, WebIm ageFX 57
ASP
 file upload 474
 integrating with eWebEditPro 739
ASP.NET, integration with eWebEditPro 744
astyle attribute
 XML feature 685
attribute types, configuration file 327
attributes
 custom tag, ColdFusion 758
 property 142
 removing from file globally 401
values
 retrieving 674
XML, see XML features, attributes
AttributeValueDefault method, XML Data object 58
autoclean
 attribute of standard element 355
AutoFillin property, XML interface object 136
autoInstallExpected, JavaScript object method 59
automatic image upload 520
 installing 522
Automatic Upload
 ASP example 559
 ASP sample database 522
 Automatic Upload Object 562
 ColdFusion example 557
controlling program
 atically 562
data island for return data 534
displaying progress information to user 500
information components 525
installing 522
Media File Object Properties 562

- Methods
 GetFieldValue 85
 GetFileDescription 87
 GetFileStatus 87
 SetFileStatus 122
modules that enable 522
overview 520
receiving a file 530
ServerName Property 146
TransferMethod Property 146
XML element descriptions 551
- Automatic Upload Object 562
 AllowUpload 148
 properties
 content type 148
 ContentDescription 148
 contentTitle 148
 port 149
autoupload element
 openaccess attribute 499
 resplvl attribute 500
 uploadonsave attribute 500
- AvailableSchemas method, XML interface object 60
- B**
- background color
 setting for editor 159
 text, command for 200
- bar element, configuration file 332
- BaseUrl
 ActiveX control property 166
 image selection object
 property 150
- binary, saving unicode characters as 419
- blocking tags 687
- body of document
 letting users view 391
- bodyStyle, ActiveX control property 159
- BodyStyle, ActiveX style sheet method 61
- bold, applying to text, command for 200
- bookmark, applying to text, command for 200
- boolean attribute type 327
- BorderSize, image selection object
 property 150
- browser
 requirements 224
 hardware 225
 viewing 224
 Web server 224
- BuildErrorDescEng method, XML interface object 61
- bullet, applying to text, command for 200
- button element, configuration file 333
- buttons
- adding separator bar between 241
adding space between 241
adding to menu 237
assigning images to in configuration file 360
caption text
 aligning 243
 displaying 243
images
 changing 242
 creating custom 369
rearranging on menu 240
removing from menu 240
translating to foreign language 244
- buttonTag, parameters object
 property 167
- C**
- CanInsert method, XML Data object 61
- caption
 element of XML feature 692
 menu, editing 236
- carriage return, processing when content is saved 395
- cascading style sheets, see style sheets
- Cell Properties dialog
 customizing 372
- cell properties dialog
 opening, command for 207
- cells
 deleting, command for 207
 inserting, command for 207
 merging, command for 207
 splitting, command for 207
- center justification, applying to text, command for 200
- characters
 encoding in the Value attribute 771
 special and extended, see encoding special characters
- charencode Attribute 419
- binary 420
 charref 421
 entityname 421
 latin 422
 special 422
 tips for choosing 422
 UTF-8 420
- charset, specifying for a page 160
- class
 attributes
 Microsoft Word, removing 358
- style
 apply to selected text 439
- applying to text surrounded by blocking tags 434
applying two style classes to same content 436
determining contents of dropdown list 440
determining names in dropdown list 441
removing from content 401
resolving overlapping attributes 442
suppressing contents of dropdown list 442
suppressing from dropdown list 442
translating to foreign language 441
types 440
- tags, removing from Microsoft WORD 2000 content 433
- clean dialog, displaying 396
- clean feature attributes
 charencode 395
 cr 395
 feedbacklevel 396
 hideobject 397
 If 397
 mswordfilter 397
 preferfont 397
 preservechars 398
 prompt 398
 reducetags 398
 showdonemsg 399
 showonsize 399
- cleanHTML 393
 publishing option 391
 suppressing clean message 399
- ClearStylesFromTags, ActiveX style sheet method 63
- client installation
 failure 299
- file
 directory path to 168
- pages
 customizing 298
 deleting 299
 user cancellation 299
- clientInstall, parameters object
 property 168
- cmd element, configuration file 339
- cmdbold 200
- cmdbookmark 200
- cmdcustapplytag, XML command 707
- cmdcusthide, XML command 708
- cmdcusthidetags, XML command 707
- cmdcustshow, XML command 707
- cmdcusttagattrs, XML command 707

cmdcusttaginsert, XML command 705
cmdcusttagprop, XML command 705
cmddatadesign 603
cmdfindreplace
 arguments 210
 arguments of cmdfind and cmdfindnext commands 212
 flags 211
 including slash in search string 212
 learning how many times search string was replaced 212
cmdfueditimage command 579
cmdmfpuploadall command 523
cmdmfpuploadcontent command 563
cmdmsword 29, 411
cmdprint 204
ColdFusion
 custom tags 758
 file upload 484
 parameters
 object,properties,bodyStyle 760
color
 background, command for 200
colors
 background, command for 200
 editing list on toolbar 201, 207
 restricting user access 201, 207
cols, parameters object property 168
columns
 deleting, command for 207
 inserting, command for 207
command object interface
 method
 AddItem 52
 Clear 62
 CmdFirst 63
 CmdNext 63
 FirstCommand 82
 GetProperty 90
 GetProperty String 91
 GetPropertyBoolean 91
 GetPropertyInteger 91
 ListCommandName 101
 NextCommand 108
 SetProperty 54, 55, 124
property
 CmdCaption 134
 CmdData 134
 CmdGray 134
 CmdSorted 134
 CmdText 135
 CmdToggledOn 135
 CmdToolTipText 135
 CmdType 135
 CmdVisible 135
commands 195

custom 215
data designer 603
executing via JavaScript 197
guidelines for using 199
modifying by scripting 258
removing from context menu 256
standard
 cmdabout 200
 cmdabspos 200
 cmdbackcolor 200
 cmdbackcolorvalue 200
 cmdbold 200
 cmdbookmark 200
 cmdbullet 200
 cmdcelleproperties 207
 cmdcenter 200
 cmdchr 200
 cmdchr160 200
 cmdchr169 200
 cmdchr174 200
 cmdclean 200
 cmdcopy 200, 593
 cmdcut 200
 cmddelete 201, 593
 cmddeletecells 207
 cmddeletecolumns 207
 cmddeleterows 207
 cmdextchars 201
 cmdfind 201
 cmdfindnext 201
 cmdfindreplace 201
 cmdfontcolor 201
 cmdfontcolorvalue 201, 207
 cmdfontname 201
 cmdfontsize 201
 cmdheaderlevel 202
 cmdhr 202
 cmdhyperlink 202
 cmdimage 203
 cmdindentleft 202
 cmdindentright 202
 cmdinsertcell 207
 cmdinsertcolumn 207
 cmdinsertrow 207
 cmdinserttable 207
 cmditalic 202
 cmdjustify 202
 cmdleft 202
 cmdlrbk 202
 cmdlredit 206
 cmdmergecells 207
 cmdmfeditimage 202
 cmdmfuiinsert 203
 cmdmfpuploadall 203
 cmdmsword 203
 cmdnojustify 203
 cmdnumbered 203

cmdopen 204, 594
cmdpaste 204, 594
cmdpicture 204
cmdprint 204
cmdprop 204
cmdredo 204
cmdremoveformat 204
cmdright 204
cmdrtlblk 204
cmdrtredit 204
cmdsav 204, 594
cmdselectall 205
cmdselectnone 205
cmdselstyle 205
cmdshowborders 205
cmdshowdetails 205
cmdspellcheck 205
cmdsplitcell 207
cmdstrike 205
cmdsub 205
cmdsupserscript 205
cmdtable 207
cmdtableproperties 207
cmdunderline 205
cmdundo 205, 595
cmdunlink 202, 205
cmdviewashtml 205
cmdviewasproperties 206
cmdviewaswysiwyg 206
cmdzabovetext 206
cmdzback 206
cmdzbackward 206
cmdzbelowtext 206
cmdzforward 206
cmdzfront 206
cmdzorder 206
js508table 206
jscomment 206
jshyperlink 206
jstm 206
tables 207
toolbarreset 303

comment
 custom tag, inserting 710
config element, configuration file 340
Config, ActiveX control property 160
config.xml file see configuration file
config.xsd file 663
configuration data
 WebImageFX 581
configuration file
 allowCustomize 315
 attribute types 327
 button 333
 changing location 314
 clean 393

command 195
config 326
customizing 329
 overriding user customization 317
 preventing users from 316
editHTML 392
external 386, 388
features element 327
fixing when changes have no effect 317
interface element 326
managing 312
math expression editor 409
mediafiles feature 493
 defsource element 506
 domain element 503
 maxsizek element 496
 mediaconfig element 496
 mediafiles element 495
 password element 502
 port element 506
 proxyserver element 503
 transport element 497
 username element 502
validext 495
webroot element 505
xferdir element 504
menu 349
overview 319
spellcheck 404
standard element 354
standard elements 329
 bar 332
 button 333
 cmd 339
 command 336, 339
 config 340
 features 341
 image 342
 interface 343
 listchoice 345
 menu 349
 popup 351
 selection 352
 space 353
 tooltiptext 358
style element 357
tables, managing 371
using to customize toolbar 230
viewas 390
content
 command to enable printing 204
 determining if changed 95
 estimating size 79
 loading
 from HTML file 204, 594
 into editor 773
maximum size 168
publishing options 391
read only 170
saving
 in editor 774
 to HTML file 204, 594
validating
 using a schema 664, 666
viewing as HTML 205
viewing as WYSIWYG 206
Content property, XML Data object 143
content type, Automatic Upload
 Property 148
content upload 563
 cmddmfpuploadcontent comm and 563
 determining if content changed prior to 566
 determining where content is stored 568
 enabling in the user interface 563
fields in the posted form 565
interface object properties 564
JavaScript example 565
object interface properties 564
receiving page 568
retrieving content from eWebEditPro 563
SetContent method 119
types of content 570
ContentDescription, automatic upload
 property 148
contentTitle, Automatic Upload
 Property 148
context menu
 customizing 256
 displaying 345
 removing commands 256
 suppressing 257, 345
continueparagraph attribute, standard element 356
ConvertImage method, WebImageFX 65
copying text, command for 200, 593
copyright character, command for inserting 200
cr, clean feature attribute 395
create
 JavaScript object method 66
createButton, JavaScript object method 67
CreateNew method, WebImageFX 67
custom commands 215
custom features, creating 386, 388
custom Javascript function, creating 216
Custom Tag Attribute dialog 715
 controlling operation 702
custom tag element, XML feature 683
Custom Tag Properties dialog boxes 716
custom tags
 ColdFusion 758
custom toolbar buttons, disabling while viewing HTML 390
customizing eWebEditPro parameters 752
cutting text, command for 200

D

data designer
commands 603
cmddatadesign 603
cmddataentry 603
cmddsgboolean 604
cmddsgcalc 604
cmddsgcalendar 604
cmddsgchoices 604
cmddsgfieldset 603
cmddsgfilelink 604
cmddsgimageonly 604
cmddsglistcontrol 604
cmddsgplaintext 604
cmddsgprop 604
cmddsgricharea 604
cmdvalidate 604
cmdvalidayt 604
cmdvalidxsd 604
indexing fields 613
 the index information file 614
invalid documents, saving 625
support 601
 configuration changes 601
validate commands 606
validating fields 615
validation
 defining 620
 defining calculation 619
 defining custom 621
 defining error message 621
 defining schema fragment 618
 treelmg attribute 617
DataStyle property, XML Data object 143
default style sheet 431
defaultdivonenter attribute
 standard element 356
DefDestinationDir, image selection object property 150
defsource element, mediafiles feature 506
deleting text, command for 201, 593
DelimitAttributes method, XML Data object 68
Description property 144
destination, upload, specifying 504
directory path to eWebEditPro 291

disableAllStyleSheets, ActiveX style sheet method 69
Disabled, ActiveX control property 160
disableStyleSheet, ActiveX style sheet method 69
display preferences, command 206
DIV tag
 applying to text surrounded by blocking tags 434
 inserting when user presses <Enter> 356
docbusymsg attribute, standard element 357
Document is Busy dialog, controlling appearance 357
documents
 letting users view body only 391
 letting users view entire source 391
 load waiting time, warning message 356
DocumentTemplate method 70
docxml element of XML feature 696
domain element,mediafiles feature 503
Domain, image selection object
 property 151
double click command, script activation, XML tags 676
double-click element handlers
 eWebEditProDbClickElement 193
 eWebEditProDbClickHyperlink 193
 3
 eWebEditProDbClickImage 194
 eWebEditProDbClickTable 194
dropdown lists
 adding to menu 238
 creating item that generates no command 249
dstyle attribute
 XML feature 685
DTD
 custom tag definition 678
DuplicateTag method, XML interface object 70

E

EditCommandComplete event, WebImageFX 183
EditCommandStart event, WebImageFX 183
EditComplete event, WebImageFX 184
EditFile method, WebImageFX 71
EditFromHTML method, WebImageFX 72
editHTML feature, configuration file 392
editor

content
 see content
inserting as a box 769
inserting as a button 771
instance object property 174
loading content 773
naming guidelines 778
placing more than one on page 225
placing on a web page 766
popup, detecting when activated 777
saving content 774
saving content as whole XML document 646
type
 specifying 169
editor name
 JavaScript object property 177
editorGetMethod, parameters object property 182
EkFileObject API 544
EkMediaTransfer.DLL 522
EktronFileIO
 implementing image upload 478
ElementAttributes method, XML Data object 72
ElementAttributeValues method, XML Data object 73, 74
ElementChildren method, XML Data object 74
ElementsIsEmpty method, XML Data object 75
ElementMaxCount method, XML Data object 75
ElementMinCount method, XML Data object 76
elemName, 174
embedattributes, parameters object
 property 168
empty XML tag 687, 688
EnableCreation method, WebImageFX 76
EnableFormatChange method, WebImageFX 76
EnableNameChange method, WebImageFX 77
encoding
 characters in the Value attribute 771
 special characters 417
 configuring eWebEditPro 419
 displaying Asian languages 419
 preventing for certain characters 398
 unicode characters
 saving 418
viewing 418
end tag, removing 400
equivClass attribute, style tag 436
ErrorClear method, WebImageFX 78
ErrorCode property, XML interface object 137
ErrorDescription method, WebImageFX 78
ErrorFilePos property, XML interface object 137
ErrorLine property, XML interface object 137
ErrorPos property, XML interface object 137
ErrorReason property, XML interface object 138
ErrorSrcText property, XML interface object 138
ErrorURL property 138
ErrorValue method, WebImageFX 79, 80
estimateContentSize, JavaScript object method 79
estimating size of content 79
event handler functions
 eWebEditProDbClickElement 193
 eWebEditProExecCommand 191
 eWebEditProMediaSelection 192
 eWebEditProReady 191
events
 eWebEditPro events file 295
eWebEditPro
 integrating using JavaScript 766
 integration with ColdFusion 756
 path, prepending URL with 116
eWebEditPro object
 methods
 isEditor 96
eWebEditPro.js file
 eWebEditProPath 291
 including 768
eWebEditPro.lpk 779
eWebEditPro.parameters.xmlInfo 708
eWebEditProDbClickElement 193
eWebEditProDbClickHyperlink
 double-click element handler 193
eWebEditProDbClickImage
 double-click element handler 194
eWebEditProDbClickTable
 double-click element handler 194
eWebEditProDefaults.js file
 clientInstall 168
eWebEditProEvents.js file
 onDbClickElementHandler 295
 onDbClickHyperlinkHandler 295
eWebEditProExecCommand, event handler function 191

eWebEditProExecCommandHandlers
 Array 301
eWebEditProMediaSelection
 event handler function 192
ewebeditpromessages.js file
 clientInstallMessage 294
 confirmAway 293
 doneLoading 293
 doneSaving 293
 elementNotFoundMessage 293,
 294
 errorLoading 293
 installPrompt 292
 invalidFormMethodMessage 294
 loading 292
 popupButtonCaption 292
 querySave 293
 saveFailed 293
 saving 293
 sizeExceeded 293
 waitingToLoad 292
eWebEditProReady
 event handler function 191
eWebEditProUtil JavaScript Object 4,
 307
 methods
 getOpenerInstance 90
 HTMLEncode 94
 IsOpenerAvailable 99
 properties
 editorName 181
 languageCode 181
 queryArgs 181
eWepAutoSvr.dll file 522
ExecCommand method 81, 197
extended character list, command
 for 201
extended characters, see encoding
 special characters
external features
 adding 386, 388

F

features element, configuration file 341
feedbacklevel, attribute of clean elem
 ent 396
file
 open, dialog, command for
 launching 204, 594
file upload information object properties
 FW Password 152
 FWProxyServer 152
 ImageHeight 154
 ImageWidth 154
 IsLocal 154
 Password 155
 ProxyServer 155

TransferRoot 157
FileSize, image selection object
 property 151
FileTitle, image selection object
 property 151
FileType, image selection object
 property 151
find and replace
 text dialog, command for 201
find and replace command
 arguments 210
 arguments of cmdfind and cm
 dfindnext commands 212
 flags 211
 including slash in search
 string 212
 learning how many times search
 string was replaced 212
find next command 201
find text dialog, commands for 201
fmtchange element, WebImageFX 582
focus
 ActiveX method 82
 setting programatically with
 JavaScript 82
font tags, removing 401
fonts
 changing list of 247
 command for assigning color 201
 default, specifying 248
 name, specifying in configuration
 file 380
 size
 changing list of 248
 specifying in configuration file 381
 specifying in configuration file 380
form elements 388
 entering on a web page 768
FormatDelimitedAttributes method, XML
 Data object 82
formatting
 text
 removing all, command for 204
formName, instance object property 174
FTP
 file upload 472
 image selection example 466
 selecting files from server 468
functions
 custom, creating 216
FW Password, image selection object
 property 152
FWLoginName, image selection object
 property 152
FWPort, image selection object
 property 152

FWProxyServer, image selection object
 property 152
FWUse, image selection object
 property 152
FWUsePassV, image selection object
 property 152

G

Get EnablePathResolution, image
 selection object property 153
Get IsValid, image selection object
 property 153
Get ShowResolutionOverride, image
 selection object property 153
Get WDDX() As String, ActiveX control
 property
 162
Get XferType, image selection object
 property 153
GetActiveStyleSheetTitles, ActiveX style
 sheet method 83
getBodyHTML, ActiveX method 83
getBodyText, ActiveX method 84
GetContentType 565
getDocument method, retrieving XML
 documents 709
getDocument, ActiveX method 85
GetFieldValue, Automatic Upload
 Methods 85
GetFileDescription, Automatic Upload
 Methods 87
GetFileStatus, Automatic Upload
 Methods 87
getHeadHTML method, for XML docum
 ent 709
getHeadHTML, ActiveX method 88
GetImageInformation method, WebIm
 ageFX 89
getProperty, ActiveX method 90
getPropertyBoolean, ActiveX method 90
getPropertyInteger, ActiveX method 91
getPropertyString, ActiveX method 91
getSelectedHTML, ActiveX method 92
getSelectedText, ActiveX method 92
GetTagAttribute method,,XML Data
 object 92
GetValidFormats method, WebIm
 ageFX 92
GetXPath method, XML data object m
 ethods 93
given, image path resolution 486
graphics, see images

H

HandledInternally, image selection
 object property 153

hardware requirements 225
 header, XML, retrieving 141
 headings
 changing list of 248
 specifying in configuration file 382
 height, instance object property 175
 hideAboutButton, ActiveX control
 property 162
 hideobject, clean feature attribute 397
 horizontal line, command for
 inserting 202
 horizontal tags 688
 horizontal XML tag 686
 HorizontalSpacing, image selection
 object property 154
 HTML
 cleaning 393
 editing 392
 file, loading into content 204, 594
 instance object property 175
 source code
 viewing 390
 disabling custom buttons 390
 viewing content as, command
 for 205
 HTTP file upload 473
 Hyperlink dialog box
 customizing 445
 default values 452
 hyperlinks
 assigning, command for 202
 managing 445
 removing, command for 202, 205

I

Icon property 144
 icons, see images
 id, instance object property 175
 image element, configuration file 342
 image file
 dynamically selecting upload
 location 513
 image selection
 database samples 470
 examples of implementing 459
 FTP
 example 466
 requirements 467
 preventing user from upload 498
 workflow 455
 image selection object
 methods
 FileExistsLocally 81
 retrieveHTMLString 116
 UseHTMLString 131
 properties
 accessing programmatically 488

alignment 149
 AllowSubDirectories 149
 AllowUpload 150
 BaseURL 150
 BorderSize 150
 DefDestinationDir 150
 DefSourceDir 150
 Domain 151
 FileSize 151
 FileTitle 151
 FileType 151
 FWLoginName 152
 FWPort 152
 FWUse 152
 FWUsePassV 152
 Get EnablePathResolution 153
 Get IsValid 153
 Get ShowResolutionOverride 153
 Get XferType 153
 HandledInternally 153
 HorizontalSpacing 154
 LoginName 154
 MaxFileSizeK 155
 NeedConnection 155
 Port 155
 ProxyServer 156
 RemotePathFileName 156
 ResolvePath 156
 ShowHeight 157
 ShowWidth 157
 SrcFileLocationName 157
 TransferMethod 157
 TransferRoot 158
 Use PassV 158
 ValidConnection 158
 ValidExtensions 158
 VerticalSpacing 158
 WebPathName 159
 WebRoot 159
 image upload
 automatic 520
 implementing 472
 see also image selection
 imageedit element, eWebEditPro
 configuration data 580
 ImageEditor method, WebImageFX 94
 ImageError event, ImageError 184
 ImageHeight, image selection object
 property 154
 images
 absolute positioning 200
 assigning to buttons in
 configuration file 360
 changing transfer method on the
 fly 490
 creating custom 369
 entry point for external scripts 489

external 360
 formats supported 360
 inserting, loaded by external m
 echanism 491
 internal 360
 media file object, accessing
 programmatically 488
 modifying upload directory 492
 path, resolving 486
 properties
 accessing via Netscape 488
 repository, setting up 510
 setting external page param
 eters 490
 setting height 154
 setting width 154
 sources 360
 uploading content copied from
 another application 520
 ImageWidth, image selection object
 property 154
 imgcreate element, WebImageFX 583
 imgedit element, WebImageFX 583
 imgfmt element, WebImageFX 584
 including the eWebEditPro Javascript
 file 768
 indent
 left, command 202
 right, command 202
 indexing data designer fields 613
 initialize toolbar event 189
 Insert Custom Tag dialog 714
 controlling operation 701
 Insert Hyperlink dialog
 specifying default values 452
 Insert Table dialog
 customizing 372
 insertMediaFile, instance object m
 ethod 94
 installation pages, client, custom
 izing 298
 installPopup, JavaScript object
 property 178
 InstallPopupQuery, parameters object
 property 173
 installPopUpurl, parameters object
 property 172
 instance object 304
 events
 onerror event 305
 methods
 insertMediaFile 94
 isChanged 95
 isEditor 97
 load 104
 save 116
 properties

editor 174
elemName 174
formName 174
height 175
html 175
id 175
maxContentSize 175
name 176
receivedEvent 176
status 176
type 176
width 177
instances collection, JavaScript object
 property 178
instanceTypes array 305
integer attribute type 328
integrating eWebEditPro
 using JavaScript 766
 with ASP 739
 with ASP.NET 744
 with ColdFusion 756
 with JSP 735
 with PHP 762
interface element, configuration file 343
interface object (XML) 719
interface, user
 defining 329
invisible elements in content, show 205
IsAttributeRequired method, XML Data
 object 95
isAutoInstallSupported, JavaScript
 object property 178
isChanged field 566
isChanged, JavaScript object method 95
IsDataField method, XML data object m
 ethods 96
IsDirty method, WebImageFX 96
IsDirty, ActiveX control property 162
isEditor
 ewebeditpro object method 96
 instance object method 97
isEditorReady() As Boolean, ActiveX m
 ethod 97
isInstalled, JavaScript object
 property 179
isLocal, image selection object
 property 154
IsPresent method, WebImageFX 99
isSupported, JavaScript object
 property 179
IsTagApplied method 99
IsValid method, XML Data object 100
isVisible method, WebImageFX 101
italic, command for applying to text 202

J

JavaScript

custom function, creating 216
files, customizing 291
object model 300
using to execute commands 197
JavaScript object 300
 events
 onbeforeedit 188
 onbeforeload 189
 onbeforesave 189
 oncreate 187
 oncreatebutton 188
 onedit 188
 onerror 190
 onload 190
 onready 190
 onsave 189
 ontoolbarreset 189
eWebEditProUtil 4, 307
methods
 addEventHandler 50
 autoInstallExpected 59
 create 66
 createButton 67
 edit
 JavaScript object method 71
 estimateContentSize 79
 getDocument() for XML docum
 ent 709
 getHeadHTML() for XML docum
 ent 709
 isChanged 95
 load 105
 openDialog 108, 109
 outerXML 109
 refreshStatus 113
 resolvePath(url) 116
 save 117
 setDocument for XML docum
 ent 710
 setHeadHTML for XML docum
 ent 710
 XMLProcessor() As XML
 Object 709
properties 300
 actionOnUnload 177
 editor name 177
 installPopup 178
 instances collection 178
 isAutoInstallSupported 178
 isInstalled 179
 isSupported 179
 parametersobject 179
 status 179
 EWEP_STATUS_FATALERR
 OR 180
 EWEP_STATUS_INSTALLED
 179

EWEP_STATUS_LOADED 17
 9
EWEP_STATUS_LOADING 1
 79
EWEP_STATUS_NOTINSTAL
 LED 180
EWEP_STATUS_NOTLOADDE
 D 179
EWEP_STATUS_NOTSUPPO
 RTED 179
EWEP_STATUS_SAVED 179
EWEP_STATUS_SAVING 179
EWEP_STATUS_SIZEEXCEE
 DED 180
EWEP_STATUS_UNABLETO
 SAVE 180
upgradeNeeded 180
version 180
XmlInfo 709
js508table command 206
jscomment command 206
 implementing 710
jshyperlink command 206
JSP, integration with eWebEditPro 735
jstm command 206

L

language
 displaying menus and dialogs in
 non-European language 285
editor
 modifying 265
eWebEditPro screens and menus,
 changing 265
foreign
 spell checking 405
 spell checking foreign 287
left justification, command for
 applying 202
left-to-right orientation, command for
 setting 202
license information, where stored 779
License, ActiveX control property
 162
linebreak, replace paragraph tag 356
linefeed character, processing when
 content is saved 397
links, quick, see quick links
listchoice element, configuration file 345
ListCommandName method 101
ListFilesWithStatus, Media File Object
 Methods 101
load
 instance object method 104
 JavaScript object method 105
LoadedFileName method, Weblm
 ageFX 62, 105

LoadingImage event, ImageError 185
loadsch element of XML feature 699
local, image path resolution 486
locale method, parameters object 106
Locale, ActiveX control property 162
localization files 266
LoginName, image selection object
 property 154
LPK file 779

M

math expression editor
 enabling 409
maxContentSize, parameters object
 property 168
MaxFileSizeK, image selection object
 property 155
maximum content size 168
maxloadsec attribute, standard elem
 ent 356
maxsizek element, mediafiles
 feature 496
media file object
 using 514
Media File Object Methods
 AddFileForUpload 51
 AddNamedData 54
 ListFilesWithStatus 101
 ReadNamedData 112
 ReadUploadResponse 112, 113
 RemoveFieldValue 114
 RemoveFileForUpload 114
 RemoveNamedData 115
 UploadConfirmMsg 131
Media File Object Properties 562
 accessing programatically 488
 accessing with Netscape 488
 changing transfer method on the
 fly 490
 modifying upload directory 492
 setting external page param
 eters 490
 specifying image 491
 using external scripts 489
mediaconfig element, mediafiles
 feature 496
MediaFile() As Media File Object,
 ActiveX methods 106
mediafiles element, mediafiles
 feature 495
mediafiles feature
 defsource element 506
 domain element 503
 maxsizek element 496
 mediaconfig element 496
 mediafiles element 495
 password element 502

port element 506
proxyserver element 503
transport element 497
username element 502
validext 495
webroot element 505
xferdir element 504
menu element, configuration file 349
menus
 adding
 buttons 237
 dropdown list 238
 separator bar between 241
 space between buttons 241
 to toolbar 233
 aligning button caption text 243
 caption, editing 236
 changing image on buttons 242
 context, see context menu
 creating 233
 defining 230
 displaying button caption text 243
 modifying by scripting 258
 object interface 258
 placing on row with another m
 enu 235
 popup
 creating 245
 rearranging buttons 240
 removing buttons 240
 removing from toolbar 234
 right-click, see context menu
 tables,customizing 375
 toolbar
 tables,customizing 376
 translating buttons to foreign
 language 244
 user customization 315
 wrapping to new toolbar row 235
menus interface
method
 CommandAdd 64
 CommandDelete 64, 65
 CommandItem 65
 HideAbout 65, 93
 HideAllMenus 93
 PopupMenu 111
 SeparatorBarAdd 118
 SeparatorSpaceAdd 118
 ShowAbout 126
 ShowAllMenus 127
 ToolbarAdd 129
 ToolbarModify 129
Method
 SetFieldValue 121
methods
 ActiveX 310

Microsoft Office 2000 content
 cleaning 393
 preparing for copying to
 eWebEditPro 355
 removing class and style tags 433
 suppress clean message 398
Microsoft Word
 class attributes, removing 358
 editing using 29, 411
 editing XML documents 415
 initial view format 412
 options for processing
 content 412
 startupmode 412
MinChildElementCount method, XML
 Data object 107
minimal, publishing option 391
minimum size needed to show clean
 HTML dialog box 399
mswordfilter attribute 397
mycomment button, implementing 710

N

name, instance object property 176
namechange element, WebIm
 ageFX 584
namespace
 using with custom tags 726
NeedConnection, image selection object
 property 155
Netscape
 accessing media file object
 properties 488
 attributes to embed tag 168
 browser for editing 224
 browser for viewing 224
 browser support for UTF-8 428
 criteria for choosing charencode
 value 424
 maximum size of content 168
 message when user opens
 page 294
 method to provide compatibility 90
 property that indicates Esker
 plug-in installed 179
 viewing special characters 418
 writing to ActiveX control
 property 124
New Hyperlink dialog, editing quick
 links 453
non-blocking tags 687
non-breaking space character, comm
 and for inserting 200
non-empty XML tag 687, 688
numbers, applying to text, command
 for 203

O

object
 model, JavaScript [300](#)
object tag
 definition [397](#)
 hiding [397](#)
objectattributes, parameters object
 property [169](#)
onbeforeedit, JavaScript object
 event [188](#)
onbeforeload, JavaScript object
 event [189](#)
onbeforesave, JavaScript object
 event [189](#)
onblur
 ActiveX event [187](#)
oncreate, JavaScript object event [187](#)
oncreatebutton, JavaScript object
 event [188](#)
ondblclickelement
 ActiveX event [186](#)
onedit, JavaScript object event [188](#)
onerror
 event, instance object [305](#)
 JavaScript object event [190](#)
onexeccommand, ActiveX event [186](#)
onfocus
 ActiveX event [187](#)
onload, JavaScript object event [190](#)
onready, JavaScript object event [190](#)
onsave, JavaScript object event [189](#)
ontoolbarreset, JavaScript object
 event [189](#)
open file dialog, command for
 launching [204, 594](#)
openaccess attribute, autoupload elem
 ent [499](#)
openDialog, JavaScript object m
 ethod [108, 109](#)
operating system requirements,
 server [224](#)
operations element, WebImageFX [585](#)
outerXML, JavaScript object m
 ethod [109](#)

P

P tags
 removing [356](#)
page, Web, ASP, adding eWebeditPro
 to [739](#)
paragraph tag, replace with
 linebreak [356](#)
parameters
 customizing [752](#)
parameters object [306](#)
 JavaScript object property [179](#)

methods

 preferredType [169](#)
 relocate(frameName) [113](#)
 reset [116](#)

properties

 BaseUrl [166](#)
 bodyStyle [159](#)
 buttonTag [167](#)
 charset [160](#)
 clientInstall [168](#)
 cols [168](#)
 config [160](#)
 editorGetMethod [182](#)
 embedAttributes [168](#)
 Get WDDX() As String [162](#)
 hideaboutbutton [162](#)
 InstallPopupQuery [173](#)
 installPopupurl [172](#)
 maxContentSize [168](#)
 objectattributes [169](#)
 path [169](#)
 popup [171](#)
 popup.query [173](#)
 popupURL [173](#)
 popupWindowFeatures [173](#)
 popupWindowName [174](#)
 readOnly [170](#)
 rows [170](#)
 textareaAttributes [170](#)
 title [163](#)

parameters object property

 password element, mediafiles
 feature [502](#)

 Password, image selection object

 property [155](#)

 pasteHTML, ActiveX methods [109](#)

 pasteText, ActiveX methods [110](#)

 pasting text

 command [204, 594](#)

 without HTML tags [204](#)

 path

 images, resolving [486](#)

 parameters object property [169](#)

 to eWebEditPro [291](#)

 PHP, integration with eWebEditPro [762](#)

 picture dialog box, command for

 launching [204](#)

 Picture Properties dialog

 alignment field

 removing [458](#)

 setting default response [458](#)

 Picture Properties dialog

 alignment field

 modifying responses [457](#)

 pictures, see images

 placeholder

 editing properties [752](#)

plug-in

 Esker ActiveX

 impact on accessing Media File
 object [488](#)

 indicating installation of [179](#)

 PopulateTagsWithStyles, ActiveX style
 sheet method [110](#)

 popup button

 customizing [253](#)

 popup editor, detecting when
 activated [777](#)

 popup element, configuration file [351](#)

 popup menu

 creating [245](#)

 popup query

 specifying web page of [173](#)

 popup windows

 determining how many are open [8](#)

 features [173](#)

 specifying name [174](#)

 specifying web page of [173](#)

 popup, parameters object property [171](#)

 popup.query, parameters object

 property [173](#)

 popupURL, parameters object

 property [173](#)

 port element, mediafiles feature [506](#)

 port, Automatic Upload Property [149](#)

 Port, image selection object

 property [155](#)

 positioning, absolute, image or table [200](#)

 preferredType(), parameters object m
 ethod [169](#)

 presentation XSLT, controlling [605](#)

 preservewordclasses [413, 433](#)

 preservewordstyles attribute [433](#)

 print, command to enable [204](#)

 prompt attribute, clean element [398](#)

 properties

 dialog, displaying for objects such
 as Flash [204](#)

 image selection object
 488

 JavaScript object [300](#)

 parameters object [306](#)

 proxyserver element, mediafiles

 feature [503](#)

 ProxyServer, image selection object

 property [156](#)

 publish, attribute of standard elem

 ent [355](#)

 PublishHTML method, WebIm

 ageFX [111](#)

 publishing options [391](#)

 publishStyles attribute [434](#)

 publishviewassource attribute, standard
 element [356](#)

Q

quick links
 creating dynamically 454
 editing list 446
 in New Hyperlink dialog
 adding 454
 editing 453
 removing 454

R

read only content, assigning 170
ReadOnly, ActiveX control property 162
ReadResponseHeader method 112
ReadUploadResponse, Media File Object Methods 112, 113
receivedEvent, instance object property 176
redisplay toolbar command 304
redo, command for executing 204
refreshStatus, JavaScript object method 113
relocate(frameName)
 parameters object 113
RemotePathFileName, image selection object property 156
RemoveFieldValue, Media File Object Methods 114
RemoveFileForUpload, Media File Object Methods 114
reset
 parameters object 116
reset toolbar
 command 303
 event 189
resolvePath(url)
 JavaScript object method 116
ResolvePath, image selection object property 156
resplv1 attribute, autoupload element 500
repository, image, setting up 510
right justifying text, command for 204
right-click menu, see context menu
right-to-left orientation, command for setting 204
root element of XML document, displaying 698
root tag
 displaying to user 139
 retrieving 139
 setting 139
RootTag property
 XML interface object 139
rows
 deleting, command for 207
 inserting, command for 207

parameters object property 170
S
sample pages, using 735
save
 instance object method 116
 JavaScript object method 117
method
 detecting when invoked 774
 terminating 775
Save method, WebImageFX 116
SaveAs method, WebImageFX 117
SavedFileName method, WebImageFX 117
saving content 774
 body only 182
 entire HTML document 182
preventing
 when submit button is pressed 775
 when Web page unloaded 775
to HTML file 204, 594
SavingImage event, ImageError 185
SchemaExists method, XML interface object 118
schemas
 active, selecting 136
 allowing client to check if loaded 118
detecting loaded 55
error
 code 137
 description, retrieving 61
 file position 137
 line 137
 position 137
 reason 138
 source text 138
extracting element information 669
files included with eWebEditPro+XML 663
loading 54, 666
retrieving all loaded 60
selecting 668
using 666
validating XML content 664
scrollbar
 moving to left side of editor window 206
 moving to right side of editor window 204
section 508 table properties, command for inserting 206
selecting all content, command for 205
selection element, configuration file 352

server, operating system requirements 224
ServerName Property, Automatic Upload 146
setBodyHTML, ActiveX methods 119
SetConfig method, WebImageFX 119
SetContent method 119
setDocument
 ActiveX method 120
 method 710
SetFieldValue method 121
SetFileStatus, Automatic Upload Methods 122
setHeadHTML
 123
 method 710
SetLocale method, WebImageFX 123
setProperty, ActiveX method 124
SetTagAttribute method, XML Data object 125
SetValidFormats method, WebImageFX 125
shifttenter, attribute of standard element 356
show invisible elements, command 205
ShowActiveStylesDetails, ActiveX style sheet method 126
showdlg attribute, autoupload element
 autoupload element, showdlg attribute 500
showdonemsg attribute, clean element 399
ShowHeight, image selection object property 157
ShowIcon property 144
showlistonsave attribute, autoupload element
 autoupload element,
 showlistonsave attribute 501
ShowName property, XML Data object 144
showonsize attribute, clean element 399
ShowRootTag property 139
ShowWidth, image selection object property 157
Simtag element of XML feature 695
Simtaglist element of XML feature 694
size of content, estimating 79
space element
 configuration file 353
span tags
 applying to text surrounded by blocking tags 434
 with font styles, converting to font tags 397
special characters
 command for inserting 200

commands 212
see also encoding special characters
spell check
 manual, command 205
spell checker
 as you type 407
 adjusting speed 407
enabling 404
foreign language 287, 405
image that indicates misspelled word 407
specifying number of replacement words 407
using without Microsoft Word 406
SrcFileName, image selection object property 157
srcName
 event object property 167
srcPath, ActiveX control property 163
standard command, detecting when executed 222
standard element
 configuration file 354
startupmode, MSWord attribute 412
status
 instance object property 176
 JavaScript object property 179
strikethrough, applying to text, command for 205
string attribute type 328
style
 class see class, style
style attribute
 XML feature 685
style element
 configuration file 357
Style Sheet, ActiveX control property 163
style sheets 430
 applying 431
 applying a style class to selected text 439
 default 431
publishStyle attribute's effect 433
replacing default 431
specifying
 for a page 432
 for one editor 433
 in config.xml 432
suppressing style classes from dropdown menu list 442
three levels 431
translating style classes 441
style tags, removing from Microsoft WORD 2000 content 433

subscript, applying to text, command for 205
superscript, applying to text, command for 205
system requirements 224

T

table borders, command for displaying 205
table properties dialog
 opening, command for 207
tables
 absolute positioning 200
 commands 207
 customizing table dialogs 372
 enabling in the configuration file 372
 inserting, command for 207
 menu, customizing 375
 toolbar menu, customizing 376
Tag Properties dialog, controlling operation 703
tagattrdlg element of XML feature 702
TagCount method 127
tagdefault element of XML feature 693
tagdefinitions element of XML feature 683
tagelement element 401
taginsdlg element of XML feature 701
TagName property, XML Data object 145
tagonly element 401
tagpropdlg element of XML feature 703
tags
 active, retrieving 50
 blocking 687
 custom, ColdFusion 758
 empty, XML 688
 horizontal 688
 HTML
 removing 401
 removing content between 401
 non-blocking 687
 nonempty 688
 removing unnecessary 398
 removing, with no attributes 402
 vertical 688
 XML, see XML tags
Tagspec element, XML feature 684
TagStyle property, XML Data object 145
TagType property, XML Data object 145
tagWoAttr element 402
target frame list, hyperlink dialog, editing 450
text
 formatting, command for removing all 204
 pasting without HTML tags 204

See Also content
textareaAttributes, parameters object property 170
Thumbnail method, WebImageFX 127
title, HTML page, setting 163
toolbarreset command 303
toolbars
 defining 230
 initialization event 189
 preventing user customization 316
 reacting to the creation of 303
 redisplaying 304
 reset event 189
 resetting 303
 user customization 315
Toolbars() As Toolbar Control Object, ActiveX method 130
tooltiptext element 358
trademark character, command for inserting 200, 206
TransferMethod Property, Automatic Upload 146
TransferMethod, image selection object property 157
TransferRoot, image selection object property 158
transform element of XML feature 698
Transform method, XML interface object 130
TransformOnLoad property, XML interface object 140
TransformOnSave property, XML interface object 140
translating eWebEditPro to another language 266
transport element, mediafiles feature 497
type
 event object property 167
 instance object property 176
type list, hyperlink dialog, editing 448

U

underline, command 205
undo command 205, 595
unicode characters 419
 saving 418
 saving as binary 419
 saving as UTF-8 419
 viewing 418
 viewing as character references 391
upgradeNeeded, JavaScript object property 180
Upload
 Automatic, see Automatic Upload
upload
 destination, specifying 504

- destinations
 selecting dynamically 513
- directory
 selecting dynamically 513
- image, automatic 520
- UploadConfirmMsg method 131
- UploadConfirmMsg, Media File Object
 Methods 131
- uploadonsave attribute, autoupload element 500
- Use PassV, image selection object
 property 158
- user interface, defining 329
- username element,mediafiles feature 502
- UTF-8
 encoding 419, 420
 implementing a web site using 427
 saving unicode characters as 419
- ## V
- valformats element, WebImageFX 586
- Validate method, XML interface object 132
- validating
 data designer fields 615
 XML content 132, 663
- validation
 HTML content
 dialogs 731
 for compliance with XHTML standards 732
 overview 727
- ValidChildElement method, XML Data object 133
- ValidConnection, image selection object
 property 158
- validext, mediafiles feature 495
- ValidExtensions, image selection object
 property 158
- valoutformats element 587
- version, JavaScript object
 properties 180
- versionInstalled, ActiveX control
 properties 163
- vertical tags 686, 688
- VerticalSpacing, image selection object
 property 158
- View As HTML
 saving content 356
- viewas feature 390
- viewing HTML source code 390
- ## W
- Web page
 ASP, adding eWebeditPro to 739
- creating 768
- Web server requirements 224
- WebImageFX
 adding toolbar button to launch 580
- color depth, specifying 590
- command for launching 202
- configuration data 581
 imgcreate element 583
- controlling 579
- displaying 579
- events
 EditCommandComplete 183
 EditCommandStart 183
 EditComplete 184
 ImageError 184
 LoadingImage 185
 SavingImage 185
- events list 592
- feature overview 577
- fmtchange element 582
- image format, specifying 590
- imgedit element 583
- imgfmt element 584
- introduction 577
- letting users change file form at 582
- letting users change file name 584
- letting users create images 583
- methods
 AskOpenFile 57
 AskSaveAs 57
 AskSelectColor 57
 ConvertImage 65
 CreateNew 67
 EditFile 71
 EditFromHTML 72
 EnableCreation 76
 EnableFormatChange 76
 EnableNameChange 77
 ErrorClear 78
 ErrorDescription 78
 ErrorValue 79, 80
 GetImageInformation 89
 GetValidFormats 92
 ImageEditor 94
 IsDirty 96
 IsPresent 99
 IsVisible 101
 LoadedFileName 62, 105
 PublishHTML 111
 Save 116
 SaveAs 117
 SavedFileName 117
 SetConfig 119
 SetLocale 123
 SetValidFormats 125
- Thumbnail 127
- methods, list 590
- namechange element 584
- object
 assigning configuration 578
 checking availability 578
 using 578
- object, retrieving 578
- operations element 585
- specifying graphic file formats 584
- valformats element 586
- WebImagerFX
 commands 593
- WebPathName, image selection object
 property 159
- webroot element,mediafiles feature 505
- WebRoot, image selection object
 property 159
- whole XML document, saving editor content as 646
- width, instance object property 177
- window
 closing without saving content 775
- Word, Microsoft, see Microsoft Word
- wrapstylewithdiv attribute 434
- WYSIWYG
 viewing content as, command for 206
- ## X
- xferdir element,mediafiles feature 504
- Xhtml
 publishing option 391
- xhtml output
 determining 355
 specifying 355
- XML
 document
 saving content as whole 646
- interface object 719
- transformation file
 loading content 699
 saving content 699
- XML attribute values
 returning valid list of 73, 74
- XML attributes
 default values 660
 delimiting 68, 673
 determining if required 95
 displaying 659
 do not display 656
 method for returning 58
 number of occurrences 660
 removing delimiter 82
 requiring 660
 retrieving 673
 retrieving value 92

returning valid list of 72
setting 673
setting value 125
user inserting 660

XML content
validating 132

XML Data object 142, 144
methods
 AttributeValueDefault 58
 CanInsert 61
 DelimitAttributes 68
 ElementAttributes 72
 ElementAttributeValues 73, 74
 ElementChildren 74
 ElementIsEmpty 75
 ElementMaxCount 75
 ElementMinCount 76
 FormatDelimitedAttributes 82
 GetTagAttribute 92
 inChildElementCount 107
 IsAttributeRequired 95
 IsValid 100
 SetTagAttribute 125
 ValidChildElement 133

properties
 AllowEdit 142
 Attributes 142
 Content 143
 DataStyle 143
 Description 144
 Icon 144
 ShowIcon 144
 ShowName 144
 TagName 145
 TagStyle 145
 TagType 145

XML data object
methods
 GetXPath 93

XML data object methods
 IsDataField 96

XML documents
editing with Microsoft Word 415
returning full 84

XML elements
determining how many child elements can occur 107
determining validity 133
finding empty element 75
maximum occurrences 75
minimum occurrences 76
retrieving child elements 74

XML feature 600
attributes
 defining appearance 685
 displaying 686
 hiding 686

best practices 721
commands 705
 cmdcustapplytag 707
 cmdcusthidetags 708
 cmdcustshow 707
 cmdcusttagattrs 707
 cmdcusttaginsert 705
 cmdcusttagprop 705

defining appearance of data section 685
defining appearance of tag section 685
defining tags
 in a script 649
 in config.xml 648

dialog boxes
 Custom Tag Attributes 715
 Custom Tag Properties 716
 Insert Custom Tags 714

DTD support 663

elements
 ashow attribute 686
 astyle attribute 685
 caption 692
 CustomTag 683
 docxml 696
 dstyle attribute 685
 loadsch 699
 Simtag 695
 Simtaglist 694
 style attribute 685
 tagattrdlg 702
 tagdefault 693
 tagdefinitions 683
 taginsdlg 701
 tagpropdlg 703
 Tagspec 684
 transform 698
 xsd 700

elements that control dialog boxes 701

loading content 644

methods
 IsTagApplied 99
 TagCount 127

properties
 eWebEditPro.parameters.xmlInfo 708

retrieving content 645, 646
sample files 648
schema support 663
storing content 646
tags see XML tags
validating content 645, 663
xmlinfo DTD definition 678

displaying root element 698
validating 230, 319

XML interface object 138, 139
methods
 ActiveTag 50
 AddSchema 54
 AnySchemasLoaded 55
 ApplyTag 56
 AvailableSchemas 60
 BuildErrorDescEng 61
 DuplicateTag 70
 SchemaExists 118
 Transform 130
 Validate 132

properties
 ActiveSchema 136
 AutoFillin 136
 ErrorCode property 137
 ErrorFilePos property 137
 ErrorLine 137
 ErrorPos property 137
 ErrorReason 138
 ErrorSrcText 138
 ErrorURL 138
 RootTag 139
 ShowRootTag 139
 TransformOnLoad 140
 TransformOnSave 140
 XmlHeader 141

XML object
methods
 FindDataField 81

XML tags
allowing user to edit contents 142
applying object information to current tag 56
assigning 721
assigning configuration information 649
assigning external XML stream 723
attributes
 delimiting 673
 retrieving 674

blocking vs. non-blocking 725
browser limitations 724
changing 672
creating duplicate 70
data width 724
default configuration 721
defining in configuration data 722
detecting ability to insert 61
determining display 650
determining if selected 674
determining if valid 100
determining tag name 674

determining whether name appears 144	parameters that determine tag style 145	ActiveX control property 164
displaying description 144	path to glyph 144	attribute 649
double click notification 676	tag 143	xmlinfo.dtd 678
editing externally 672	tag type 145	xmlinfo.xml file 663
external	script activation of double click command 676	xmlinfo.xsd file 663
editing 672	setting	XMLProcessor() As XML Object, ActiveX method 133
external tag functionality 676	attribute value 125	XPath references 628
how developers insert 653	attributes 673	xsd element of XML feature 700
how users insert 652	content 143	XSLT file
improper formatting 655	types 686	loading content 699
invisible 721	user moving 651	saving content 699
look suggestions 725	user selectable looks 722	used with getDocument method 140
modifying contents 675	using color 723	used with setDocument method 140
read only 142	where to define 648	XSLT transformation
retrieving 672	where to define appearance 648	performing on document 130
attribute value 92	XML, Data Designer, see data designer	xsltoutput element 605
attributes 673	XmlHeader property, XML interface object 141	
name 145	xmlInfo	
parameters that determine data style 143		