

1. Welcome	8
1.1 EditLive! 9 Upgrade Guide	9
1.2 Quick Start Guide	15
1.2.1 Deploying EditLive! on a Web Server	16
1.2.2 Integrating EditLive! in 3 Lines	17
1.2.3 Creating Editable Sections on a Page	19
1.2.4 Creating a Minimal Editing UI with In-place Editing	22
1.3 Install Guide	30
1.3.1 System Requirements	31
1.3.2 General Server Install Instructions	32
1.3.3 Installation Guide - JavaScript	34
1.3.4 Installation Guide - IIS Install	35
1.3.5 Installation Guide - ASP.NET	38
1.3.6 Deploying to an External Web Server	41
1.3.7 Installing TinyMCE	42
1.3.8 Client Install	43
1.3.9 Applet Security - Deployment Ruleset	44
1.3.10 Java SWT Framework	49
1.4 Developer Guide	50
1.4.1 Instantiating, Configuring, and Using EditLive!	52
1.4.1.1 Instantiating, Configuring, and Using the Applet	53
1.4.1.1.1 Instantiating the Applet	54
1.4.1.1.2 Default Values of Load Time Properties	56
1.4.1.1.3 Setting EditLive! Contents	57
1.4.1.1.4 Retrieving Content From EditLive!	58
1.4.1.1.5 Advanced APIs	62
1.4.1.1.6 Using Inline Editing	74
1.4.1.1.7 Using TinyMCE	83
1.4.1.2 Instantiating, Configuring, and Using the Java Swing SDK	88
1.4.1.2.1 Instantiating EditLive! for Java Swing in a Java Application	89
1.4.1.2.2 Java Swing APIs	91
1.4.1.2.3 Compiling Applications	92
1.4.1.2.4 Setting EditLive! For Java Swing Content	93
1.4.1.2.5 Retrieving Content From EditLive! for Java Swing	94
1.4.1.2.6 Java runtime settings	96
1.4.1.3 Encoding Content for Use with EditLive!	97
1.4.1.4 Manually Editing Configuration Files	99
1.4.1.5 Licensing EditLive!	112
1.4.1.6 Autosave	114
1.4.1.7 Automatic Hyperlinking	115
1.4.1.8 Insert HTML Fragment	116
1.4.1.9 Java Webstart for Java 9+	117
1.4.2 Enterprise Edition	118
1.4.2.1 Enabling Enterprise Edition	119
1.4.2.2 Enterprise Edition Features	120
1.4.3 Editor Appearance	122
1.4.3.1 Setting Menu and Toolbar Items	123
1.4.3.1.1 Mnemonics and Shortcuts for Menus	131
1.4.3.2 Menu and Toolbar Item List	132
1.4.3.2.1 File Commands	133
1.4.3.2.2 Edit Commands	134
1.4.3.2.3 View Commands	135
1.4.3.2.4 Insert Commands	136
1.4.3.2.5 Format Commands	137
1.4.3.2.6 Tool Commands	139
1.4.3.2.7 Table Commands	140
1.4.3.2.8 Properties Commands	141
1.4.3.2.9 Help Commands	142
1.4.3.2.10 Form Commands	143
1.4.3.2.11 Track Changes Commands	144
1.4.3.2.12 Image Editor Commands	145
1.4.3.2.13 Accessibility Commands	146
1.4.3.2.14 Broken Hyperlink Report	147
1.4.3.2.15 Commenting Commands	148
1.4.3.2.16 Equation Editor Commands	149
1.4.3.2.17 Menu and Toolbar Item Groups	150
1.4.3.3 Creating Custom Menu and Toolbar Items	151
1.4.3.4 Customizing the Color Picker	156
1.4.4 Cascading Stylesheet Support	157
1.4.4.1 Using CSS in EditLive!	158
1.4.4.1.1 Using CSS in the Applet	159
1.4.4.1.2 Using CSS in the Swing SDK	166
1.4.4.2 Using CSS Extensions to Render Custom Tags	172
1.4.4.3 Restricting what CSS classes appear in EditLive! styles drop down	176
1.4.5 Proofing and Language Tools	177
1.4.5.1 Tiny Dictionaries	178
1.4.5.2 Creating, Modifying and Adding to Dictionaries	179
1.4.5.3 Tiny Thesauruses	181
1.4.5.4 Auto Correct Spelling	182

1.4.6 Image and Media Support	183
1.4.6.1 Working with Images	184
1.4.6.1.1 Image Editing	185
1.4.6.1.2 HTTP Upload Support for Images and Objects	186
1.4.6.1.3 Drag and Drop	201
1.4.6.2 Working with Media	202
1.4.6.2.1 Using Social Media and External Media Services	204
1.4.6.2.2 Using a Media Aggregator Service	206
1.4.6.2.3 Embedding Media Using HTML5	207
1.4.6.3 Advanced Media Support	208
1.4.6.3.1 Object Tag Support	209
1.4.6.3.2 Using WebDAV with EditLive!	212
1.4.6.3.3 Enabling WebDAV on a Web Server	215
1.4.6.3.4 Image Insertion Dialog's Browser Component	217
1.4.7 Collaboration	218
1.4.7.1 Getting Started With Track Changes	219
1.4.7.2 Track Changes Serialization Format	220
1.4.7.3 Commenting	224
1.4.8 Web Content Accessibility	225
1.4.8.1 Accessibility Compliance	226
1.4.8.2 Accessibility As You Type	227
1.4.8.3 Table Accessibility	228
1.4.9 Internationalization Support	229
1.4.9.1 Internationalization Support Overview	230
1.4.9.2 Character Sets Supported	232
1.4.9.3 Specifying Character Sets for Internationalization	233
1.4.9.3.1 Specifying Character Sets in the Applet	234
1.4.9.3.2 Specifying Character Sets in the Swing SDK	236
1.4.10 Read Only Content and Custom Tags	238
1.4.10.1 Creating Read Only Content	239
1.4.10.2 Using Custom Tags	240
1.4.11 Equation Editor	242
1.4.11.1 Integrating the Tiny Equation Editor	243
1.4.12 Troubleshooting	245
1.4.12.1 Load Time Troubleshooting	246
1.4.12.1.1 Load Time Troubleshooting in the Applet	247
1.4.12.1.2 Load Time Troubleshooting in the Swing SDK	249
1.4.12.2 Run Time Troubleshooting	251
1.4.12.3 Minimizing an EditLive! Deployment	252
1.4.12.4 Optimizing Load Time	253
1.4.12.5 Managing Crashes	255
1.4.13 Plugins	257
1.4.13.1 Creating and Using Plugins in the Applet	258
1.4.13.2 Creating and Using Plugins in the Swing SDK	259
1.4.14 HTML5 Support	260
1.4.14.1 Disabling HTML5 Features	261
1.5 Reference	262
1.5.1 Load Time Methods	265
1.5.1.1 Load Time Properties Scope	266
1.5.1.2 JavaScript Constructor	267
1.5.1.3 addEditableSection Method	268
1.5.1.4 addJar Method	270
1.5.1.5 addPluginAsText Method	271
1.5.1.6 addPlugin Method	272
1.5.1.7 AppletSize Property (ASP.NET only)	273
1.5.1.8 closeOnFocusLost Method	274
1.5.1.9 Content Property (ASP.NET only)	275
1.5.1.10 EnableViewState Property (ASP.NET)	276
1.5.1.11 ID Property (ASP.NET only)	277
1.5.1.12 Init Method (ASP only)	278
1.5.1.13 InlineEditing Property (ASP.NET only)	279
1.5.1.14 InlineEditingCSS Property (ASP.NET only)	280
1.5.1.15 setAutoSubmit Method	281
1.5.1.16 setBaseURL Method	282
1.5.1.17 setBody Method	284
1.5.1.18 setConfigurationFile Method	286
1.5.1.19 setConfigurationText Method	288
1.5.1.20 setCookie Method	290
1.5.1.21 setCrashAction Method	291
1.5.1.22 setDebugLevel Method	292
1.5.1.23 setDirection Method	294
1.5.1.24 setDirectionForEditableSection Method	295
1.5.1.25 setDisplayEditableMarker Method	296
1.5.1.26 setDocument Method	297
1.5.1.27 setDownloadDirectory Method	298
1.5.1.28 setEditableSectionCSS Method	300
1.5.1.29 setExpressEdit Method	301
1.5.1.30 setFocusOnLoad Method	303
1.5.1.31 setHead Method	304

1.5.1.32	setHeight Method	305
1.5.1.33	setHideButtonIconURL Method	306
1.5.1.34	setHideButtonText Method	307
1.5.1.35	setHttpLayerManager Method	308
1.5.1.36	setJREDownloadURL Method	310
1.5.1.37	setLocalDeployment Method	311
1.5.1.38	setLocale Method	312
1.5.1.39	setMinCrashTimeout Method	314
1.5.1.40	setMinimumJREVersion Method	315
1.5.1.41	setName Method	316
1.5.1.42	setOnInitComplete Method	317
1.5.1.43	setOutputCharset Method	319
1.5.1.44	setPreload Method	321
1.5.1.45	setReadOnly Method	323
1.5.1.46	setReturnBodyOnly Method	324
1.5.1.47	setShowButtonIconURL Method	326
1.5.1.48	setShowButtonText Method	327
1.5.1.49	setShowSystemRequirementsError Method	328
1.5.1.50	setStyles Method	329
1.5.1.51	setUseLiveConnect Method	331
1.5.1.52	setUseMathML Method	332
1.5.1.53	setUserName Method	333
1.5.1.54	setWidth Method	334
1.5.1.55	show Method	335
1.5.1.56	showAsButton Method	336
1.5.1.57	showInElement Method	337
1.5.1.58	setResizableSections	338
1.5.2	Run Time Methods	339
1.5.2.1	closeActiveEditableSection Method	340
1.5.2.2	getBody Method	341
1.5.2.3	getCharCount Method	343
1.5.2.4	getContentForEditableSection Method	344
1.5.2.5	getDocument Method	346
1.5.2.6	getEditableSections Method	347
1.5.2.7	getSelectedText Method	349
1.5.2.8	getStyles Method	351
1.5.2.9	getWordCount Method	352
1.5.2.10	insertHTMLAtCursor Method	353
1.5.2.11	insertHyperlinkAtCursor Method	354
1.5.2.12	isDirty Method	356
1.5.2.13	openEditableSection Method	357
1.5.2.14	postDocument Method	358
1.5.2.15	removeEditableSection Method	359
1.5.2.16	setBackgroundMode Method	360
1.5.2.17	setBody Method (Run Time)	361
1.5.2.18	setContentForEditableSection Method	362
1.5.2.19	setDocument Method (Run Time)	364
1.5.2.20	setProperties Method	365
1.5.2.21	uploadImages Method	366
1.5.2.22	getContent Method	367
1.5.2.23	performRaiseEvent method	369
1.5.2.24	isEditableSectionDirty Method	370
1.5.2.25	setIsDirty Method	371
1.5.3	Utility Methods	372
1.5.3.1	quickStart Method	373
1.5.4	Configuration File Elements	374
1.5.4.1	accessibilityChecks	375
1.5.4.2	action	378
1.5.4.2.1	action (Applet)	379
1.5.4.2.2	action (Swing SDK)	381
1.5.4.3	authentication	384
1.5.4.4	base	385
1.5.4.5	body	386
1.5.4.6	category	388
1.5.4.7	color	389
1.5.4.8	colorPalette	390
1.5.4.9	comboBoxItem	391
1.5.4.10	customComboBoxItem	394
1.5.4.10.1	customComboBoxItem (Applet)	395
1.5.4.10.2	customComboBoxItem (Swing SDK)	398
1.5.4.11	customMenuItem	400
1.5.4.11.1	customMenuItem (Applet)	401
1.5.4.11.2	customMenuItem (Swing SDK)	405
1.5.4.12	customTags	409
1.5.4.13	customToolBarButton	410
1.5.4.13.1	customToolBarButton (Applet)	411
1.5.4.13.2	customToolBarButton (Swing SDK)	415
1.5.4.14	customToolBarComboBox	419
1.5.4.15	document	420

1.5.4.16	doubleClickActions	421
1.5.4.17	editLive	422
1.5.4.18	ephoxLicenses	424
1.5.4.19	excellImport	425
1.5.4.20	head	427
1.5.4.21	html	428
1.5.4.22	htmlFilter	429
1.5.4.23	htmlImport	431
1.5.4.24	httpUpload	433
1.5.4.25	httpUploadData	435
1.5.4.26	httpPostData	436
1.5.4.27	hyperlink	437
1.5.4.28	hyperlinkList	439
1.5.4.29	hyperlinks	440
1.5.4.30	image	441
1.5.4.31	imageBrowser	443
1.5.4.32	imageDialog	445
1.5.4.33	imageList	446
1.5.4.34	images	447
1.5.4.35	inlineToolBar	449
1.5.4.36	inlineToolbars	451
1.5.4.37	license	452
1.5.4.38	link	454
1.5.4.39	mailtoLink	456
1.5.4.40	mailtoList	457
1.5.4.41	mathml	458
1.5.4.42	mediaSettings	459
1.5.4.43	menu	460
1.5.4.44	menuBar	462
1.5.4.45	menuItem	463
1.5.4.46	menuItemGroup	465
1.5.4.47	menuSeparator	466
1.5.4.48	meta	467
1.5.4.49	multimedia	469
1.5.4.50	otherLicenses	471
1.5.4.51	param	472
1.5.4.52	placesInDocumentList	474
1.5.4.53	plugins (config)	475
1.5.4.54	realm	476
1.5.4.55	repository	478
1.5.4.56	shortcutMenu	481
1.5.4.57	shrtMenu	482
1.5.4.58	shrtMenuItem	483
1.5.4.59	shrtMenuSeparator	484
1.5.4.60	sourceEditor	485
1.5.4.61	spellCheck	486
1.5.4.61.1	spellCheck (Applet)	487
1.5.4.61.2	spellCheck (Swing SDK)	489
1.5.4.62	style	491
1.5.4.63	submenu	492
1.5.4.64	symbol	493
1.5.4.65	symbols	494
1.5.4.66	template	495
1.5.4.67	templates	497
1.5.4.68	textImport	498
1.5.4.69	thesaurus	499
1.5.4.69.1	thesaurus (Applet)	500
1.5.4.69.2	thesaurus (Swing SDK)	501
1.5.4.70	title	502
1.5.4.71	toolbar	503
1.5.4.72	toolbarButton	505
1.5.4.73	toolbarButtonGroup	506
1.5.4.74	toolbarComboBox	507
1.5.4.75	toolbars	508
1.5.4.76	toolbarSeparator	509
1.5.4.77	trackChanges	510
1.5.4.78	type	511
1.5.4.79	types	513
1.5.4.80	webdav	514
1.5.4.81	webeqLicense	516
1.5.4.82	wordImport	517
1.5.4.83	wysiwygEditor	519
1.5.4.84	services	522
1.5.4.85	service	523
1.5.4.86	contentLanguages	525
1.5.4.87	language	526
1.5.5	Plugin XML Elements	527
1.5.5.1	plugin	528
1.5.5.2	menu (plugin)	531

1.5.5.3 script	532
1.5.5.4 advancedapis	533
1.5.5.4.1 advancedapis (Applet)	534
1.5.5.4.2 advancedapis (Swing SDK)	535
1.5.6 Java API	536
1.5.7 Menu and Toolbar Items	537
1.6 Tutorials	538
1.6.1 Instantiating the Editor	540
1.6.1.1 Instantiating an EditLive! Applet	541
1.6.1.1.1 Instantiation Tutorial	542
1.6.1.1.2 Instantiation Tutorial Code	544
1.6.1.2 Instantiating EditLive! in a Swing Application	545
1.6.1.2.1 Instantiation of a Swing Application Tutorial	546
1.6.1.2.2 Instantiation of a Swing Application Code	550
1.6.2 Creating and Editing Configuration Files	552
1.6.2.1 Creating and Editing Configuration Files Tutorial	553
1.6.2.2 Specifying the Configuration File in an EditLive! Applet Tutorial	554
1.6.2.3 Specifying the Configuration File in EditLive! for Java Swing Tutorial	555
1.6.2.4 Creating and Editing Configuration Files Code	557
1.6.3 Installing a License	566
1.6.3.1 Installing a License Tutorial	567
1.6.3.2 Installing a License Code	569
1.6.4 Setting the Document	578
1.6.4.1 Setting the Document in the Applet	579
1.6.4.1.1 Setting the Document in the Applet Tutorial	580
1.6.4.1.2 Setting the Document in the Applet Code	583
1.6.4.2 Setting the Document in the Swing SDK	585
1.6.4.2.1 Setting the Document in the Swing SDK Tutorial	586
1.6.4.2.2 Setting the Document in the Swing SDK Code	594
1.6.5 Setting the Body	596
1.6.5.1 Setting the Body in the Applet	597
1.6.5.1.1 Setting the Body in the Applet Code	598
1.6.5.1.2 Setting the Body in the Applet Tutorial	600
1.6.5.2 Setting the Body in the Swing SDK	603
1.6.5.2.1 Setting the Body in the Swing SDK Tutorial	604
1.6.5.2.2 Setting the Body in the Swing SDK Code	612
1.6.6 Getting the Document	614
1.6.6.1 Getting the Document in the Applet Overview	615
1.6.6.1.1 Getting the Document in the Applet Tutorial	616
1.6.6.1.2 Getting the Document in the Applet Code	618
1.6.6.2 Getting the Document in the Swing SDK	620
1.6.6.2.1 Getting the Document in the Swing SDK Tutorial	621
1.6.6.2.2 Getting the Document in the Swing SDK Code	627
1.6.7 Getting the Body	629
1.6.7.1 Getting the Body in the Applet	630
1.6.7.1.1 Getting the Body in the Applet Tutorial	631
1.6.7.1.2 Getting the Body in the Applet Code	633
1.6.7.2 Getting the Body in the Swing SDK	635
1.6.7.2.1 Getting the Body in the Swing SDK Tutorial	636
1.6.7.2.2 Getting the Body in the Swing SDK Code	642
1.6.8 Adding and Removing Menu or Toolbar Items	644
1.6.8.1 Adding and Removing Menu or Toolbar Items Tutorial	645
1.6.8.2 Adding and Removing Menu or Toolbar Items Code	647
1.6.9 Specifying Character Set	655
1.6.9.1 Specifying Character Set in the Applet	656
1.6.9.1.1 Specifying Character Set in the Applet Tutorial	657
1.6.9.1.2 Specifying the Character Set in the Applet Code	662
1.6.9.2 Specifying Character Set in the Swing SDK	675
1.6.9.2.1 Specifying Character Set in the Swing SDK Tutorial	676
1.6.9.2.2 Specifying Character Set in the Swing SDK Code	686
1.6.10 Setting CSS	699
1.6.10.1 Setting CSS in the Applet	700
1.6.10.1.1 Setting CSS in the Applet Tutorial	701
1.6.10.1.2 Setting CSS in the Applet Code	703
1.6.10.2 Setting CSS in the Swing SDK	704
1.6.10.2.1 Setting CSS in the Swing SDK Tutorial	705
1.6.10.2.2 Setting CSS in the Swing SDK Code	708
1.6.11 Custom Toolbar Buttons	710
1.6.11.1 Custom Toolbar Buttons in the Applet	711
1.6.11.1.1 Custom Toolbar Buttons in the Applet Tutorial	712
1.6.11.1.2 Custom Toolbar Buttons in the Applet Code	715
1.6.11.2 Custom Toolbar Buttons in the Swing SDK	727
1.6.11.2.1 Custom Toolbar Button in the Swing SDK Tutorial	728
1.6.11.2.2 Custom Toolbar Button in the Swing SDK Code	737
1.6.12 Using Inline Editing (Tutorial)	750
1.6.12.1 Using Inline Editing Tutorial	751
1.6.12.2 Using Inline Editing Code	756
1.6.12.2.1 inlineEditing.html	757
1.6.12.2.2 posthandler.asp	759

1.6.13 Creating Plugins Utilizing Advanced APIs	760
1.6.13.1 Creating Plugins Utilizing Advanced APIs Tutorial	761
1.6.13.2 Creating Plugins Utilizing Advanced APIs Code	767
1.6.13.2.1 advancedAPIPlugin.html	768
1.6.13.2.2 AdvancedAPIPlugin.java	769
1.6.13.2.3 advancedAPIPlugin.xml	770
1.6.14 Capturing Content Before Submit	771
1.6.14.1 Capturing Content Before Submit Tutorial	772
1.6.14.2 Capturing Content Before Submit Code	776
1.6.15 Simple Plugin	778
1.6.15.1 Simple Plugin Tutorial	779
1.6.15.2 Simple Plugin Code	782
1.6.15.2.1 plugin.js	783
1.6.15.2.2 simplePlugin.html	784
1.6.15.2.3 simplePlugin.xml	785
1.6.16 Optimizing Load Times	786
1.6.16.1 Optimizing Load Times Tutorial	787
1.6.16.2 Optimizing Load Times Code	790
1.6.16.2.1 optimizingLoadTime.html	791
1.7 Examples	795
1.7.1 Introduction	796
1.7.2 Basic ASP Example	797
1.7.2.1 Basic ASP Example Documentation	798
1.7.2.2 Basic ASP Example Code	800
1.7.3 Basic ASP.NET Example	801
1.7.3.1 Basic ASP.NET Example Documentation	802
1.7.3.2 Basic ASP.NET Example Code	805
1.7.4 Inline Editing ASP.NET Example	806
1.7.4.1 Inline Editing in ASP.NET Example Documentation	807
1.7.4.2 Inline Editing in ASP.NET Example Code	809
1.7.5 Basic ColdFusion Example	810
1.7.5.1 Basic ColdFusion Example Documentation	811
1.7.5.2 Basic ColdFusion Example Code	813
1.7.6 Basic PHP Example	814
1.7.6.1 Basic PHP Example Documentation	815
1.7.6.2 Basic PHP Code	817
1.7.7 ColdFusion in MySQL Example	818
1.7.7.1 MySQL in ColdFusion Example Documentation	819
1.7.7.2 MySQL in ColdFusion Code	826
1.7.7.2.1 add.cfm	827
1.7.7.2.2 delete.cfm	829
1.7.7.2.3 edit.cfm	831
1.7.7.2.4 start.cfm	833
1.7.7.2.5 view.cfm	835
1.7.7.2.6 xt_add.cfm	836
1.7.7.2.7 xt_delete.cfm	837
1.7.7.2.8 xt_edit.cfm	838
1.7.8 MySQL in PHP Example	839
1.7.8.1 MySQL in PHP Example Documentation	840
1.7.8.2 MySQL in PHP Code	850
1.7.8.2.1 add.php	851
1.7.8.2.2 delete.php	853
1.7.8.2.3 edit.php	855
1.7.8.2.4 l_database.php	857
1.7.8.2.5 start.php	859
1.7.8.2.6 view.php	861
1.7.8.2.7 xt_add.php	862
1.7.8.2.8 xt_delete.php	863
1.7.8.2.9 xt_edit.php	864
1.7.9 Automated Plugin Loading PHP Example	865
1.7.9.1 Automated Plugin Loading PHP Documentation	866
1.7.9.2 Automated Plugin Loading PHP Code	870
1.7.9.2.1 example.php	871
1.7.9.2.2 pluginLoader.php	873
1.8 IBM Web Content Management Integration	874
1.8.1 IBM Web Content Management 7+	875
1.8.1.1 Installing EditLive IBM WCM 7+	876
1.8.1.2 Upgrading the Integration IBM WCM 7+	883
1.8.1.3 Additional Configuration Items	887
1.8.2 IBM Web Content Management 6.1+	888
1.8.2.1 Installing EditLive IBM WCM 6.1+	889
1.8.2.2 Upgrading the Integration IBM WCM 6.1+	895
1.8.3 IBM Web Content Management 6.1	899
1.8.3.1 Installing EditLive IBM WCM 6.1	900
1.8.3.2 Upgrading the Integration IBM WCM 6.1	907
1.8.4 Specifying Configurations in IBM WCM	911
1.8.5 Licensing EditLive IBM WCM	913
1.8.6 Troubleshooting in IBM WCM	914
1.8.7 Uninstalling the IBM WCM Integration	915

1.9 OpenText RedDot CMS Integration .....	916
1.9.1 Installing EditLive for RedDot .....	917
1.9.2 Licensing EditLive for RedDot .....	918

# Welcome

Welcome to the documentation for Ephox EditLive! version 9.



# EditLive! 9 Upgrade Guide

EditLive! 9 contains a broad range of new functionality including media functionality, HTML5 support, new user interface options and more. This document describes the major changes in EditLive! 9 and how to upgrade your existing EditLive! deployment to take advantage of them.

- [Licensing](#)
- [New Editor Features Including Toolbar Buttons and Menu Items](#)
  - [Social and Embedded Media](#)
  - [HTML5 Media - AUDIO and VIDEO Tags](#)
  - [Page Width and Mobile Preview](#)
  - [HTML5 Semantic Elements](#)
  - [Enhanced HTML Support](#)
  - [New Toolbar Megamenus](#)
- [New User Interface Options](#)
- [New Style Options](#)
- [Platform Support](#)
- [Behavioural Changes](#)
  - [Loading Behaviour](#)
  - [Updated EditLive! UI](#)
  - [List Toolbar Buttons Converted to Megamenus](#)
  - [User-friendly Style Names](#)
  - [Accessibility Checks](#)
  - [Media Preview](#)
  - [Create and Remove Section Commands](#)
  - [HTTP Clients](#)
  - [raiseEvent API Passes the Applet ID](#)
  - [WebDav Functionality Removed in 9.1](#)
  - [JavaScript APIs Removed in 9.1](#)
  - [Java APIs Removed in 9.1](#)

## Licensing

All EditLive! users will be required to upgrade their license to use EditLive! 9. An example version 9 license key looks like the following:

```
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-7C38-9CFD-AE9B"
    licensee="For Evaluation Only"
    release="9.0"
    type="Evaluation License"
    productivityPack="true"
  />
</ephoxLicenses>
```

The version number in your license key must say 9.0.

To get your EditLive! 9 license, please log a support case in the [Tiny support system](#).

## New Editor Features Including Toolbar Buttons and Menu Items

### Social and Embedded Media

EditLive! 9 includes upgraded media support, with enhanced embedding functionality and support for social media services like YouTube, Vimeo, Flickr, Google Maps, SlideShare and other web-based media aggregator services like [embed.ly](#). This functionality makes embedding social and cloud-based media as simple as creating a hyperlink. Users need only copy the URL of the resource (usually the browser's URL) into EditLive!'s media dialog and then the oEmbed standard is used to obtain the media at that URL and embed it into the content in EditLive!.

This functionality has been designed as a replacement for the process of creating an OBJECT tag - a process that's often too complex for content authors - and avoids the need to access EditLive!'s code view to paste in HTML code.

Content is inserted via the *Insert Media* dialog which supports adding content from services that support the [oEmbed specification](#) or by insertion of an arbitrary HTML fragment to represent rich media.

For more information on Social and Embedded Media support, including configuration information, refer to the Developer Guide [Using Social Media and External Media Services](#).

## HTML5 Media - AUDIO and VIDEO Tags

EditLive! 9 introduces support for insertion of HTML5 <video> and <audio> tags. This is achieved through the Video and Audio tabs on the *Insert Media* dialog with placeholder images displayed within EditLive!. This functionality works with content that's already available on web servers.

The following video formats are supported:

- MP4
- OGG
- WebM

The following audio formats are supported:

- WAV
- MP3
- OGG

For more information on HTML 5 Audio and Video support refer to the Developer Guide [Embedding Media Using HTML5](#).

Note

HTML5 media cannot be played back in EditLive!'s Design view. In this view the media will be inserted with an appropriate place holder.

Media can be played back in the Preview tab.

## Page Width and Mobile Preview

EditLive! 9 introduces the ability to render your content for a specific page width with the new Page Width function. The functionality is particularly useful when content being created by users may be read on mobile devices in addition to desktop machines. It is preconfigured with common page widths corresponding to mobile and desktop devices.

Setting the page width is via the Page Width toolbar button which provides a selection of common page widths. These include

- Smartphone Portrait
- Smartphone Landscape
- Tablet Portrait
- Tablet Landscape
- Monitor
- Widescreen Monitor
- Size to Fit
- Custom

For more information on including the Page Width on your toolbar refer to the *View Commands* section of the [Menu and Toolbar Item List](#).

## HTML5 Semantic Elements

EditLive! 9 supports the broadest range of HTML5 markup of any rich text editor. In addition to the HTML5 media tags listed previously it also supports the full range of HTML5 semantic tags and ensures the validity of any content created containing these tags. This markup provides extra semantic meaning within pages and can be particularly significant for accessibility or providing extra styling flexibility.

Users can create new content containing these tags using the Create Section command and any existing content containing this markup can now be edited by EditLive!. Designers can also create templates for use with EditLive! that include this markup.

EditLive! 9 includes rendering and editing of HTML5 semantic block elements including:

- Section
- Header
- Footer
- Article
- Aside
- Nav
- Figure
- FigCaption

HTML 5 semantic block elements can be inserted using the *Create Section* toolbar button and removed using the *Remove Section* toolbar button.

## Enhanced HTML Support

EditLive! 9 has enhanced support for the following HTML tags:

- Blockquote - This tag can now be inserted as a standalone tag or wrapped around existing content using the Blockquote option on the *Create Section* menu.
- DIV - The rendering for DIVs has been improved and they have been added as an option on the *Create Section* menu.
- iframe - EditLive! 9 provides an improved rendering for this tag using a placeholder icon and the height and width of the iframe.

## New Toolbar Megamenus







EditLive! 9 introduces megamenus that can be added to the toolbar and menus. When used on the toolbar megamenus are represented by a single toolbar icon with a chevron next to it to indicate more options are available. When used on a menu these megamenus appear as submenus.



Megamenus serve two key purposes:

- they make it easier for users to access related functionality without opening a properties dialog or a menu and;
- they enable developers to optimize the use of space on the toolbar, which can be useful when screen real estate is at a premium - with in-place editing use cases for example.

Megamenus are a collection of functionality that are represented as a single toolbar button that has an expandable menu associated with it. The items on megamenus are not customizable.

The following megamenus are available.

Menu or Tool Tip Text	Function and items	Image
Text Formatting	<p>Megamenu of formatting commands, consisting of:</p> <ul style="list-style-type: none"> <li>• Bold</li> <li>• Italic</li> <li>• Underline</li> <li>• Superscript</li> <li>• Subscript</li> <li>• Strikethrough.</li> </ul>	
Alignment and Indents	<p>Megamenu of Alignment and Indenting commands, consisting of:</p> <ul style="list-style-type: none"> <li>• Align Left</li> <li>• Align Center</li> <li>• Align Right</li> <li>• Decrease Indent</li> <li>• Increase Indent</li> </ul>	
Proofing Tools	<p>Megamenu of Proofing commands, consisting of:</p> <ul style="list-style-type: none"> <li>• Spelling</li> <li>• Disable Spell Checking as you type</li> <li>• Link Checking</li> <li>• Accessibility Report</li> <li>• Enable Accessibility as you type</li> <li>• Word Count</li> </ul>	
Help	<p>Megamenu of help and debugging commands, consisting of:</p> <ul style="list-style-type: none"> <li>• Help</li> <li>• About EditLive!</li> <li>• Enable Debug Logging</li> </ul>	
Page Width	<p>Megamenu providing access to common screen widths for mobile, tablet and desktop devices. The following widths and commands are available:</p> <ul style="list-style-type: none"> <li>• Smartphone Portrait</li> <li>• Smartphone Landscape</li> <li>• Tablet Portrait</li> <li>• Tablet Landscape</li> <li>• Monitor</li> <li>• Widescreen Monitor</li> <li>• Size to Fit - Fit to the width of the current editor instance</li> <li>• Custom - User is prompted for a custom width.</li> </ul>	
Ordered and Unordered Lists	<p>By default these megamenus insert the standard list types. Using the chevron opens a panel that enables the user to select other standard HTML list types.</p>	

Create Section	<p>Megamenu enables the user to insert or wrap content in one of the following block tags:</p> <ul style="list-style-type: none"> <li>• DIV</li> <li>• Blockquote</li> <li>• Section</li> <li>• Header</li> <li>• Footer</li> <li>• Article</li> <li>• Aside</li> <li>• Nav</li> <li>• Figure</li> <li>• FigCaption</li> </ul>	
Remove Section	<p>Megamenu enables the user to remove a blog tag within the HTML structure at the current cursor location:</p> <ul style="list-style-type: none"> <li>• DIV</li> <li>• Blockquote</li> <li>• Section</li> <li>• Header</li> <li>• Footer</li> <li>• Article</li> <li>• Aside</li> <li>• Nav</li> <li>• Figure</li> <li>• FigCaption</li> </ul> <p>Removal of the tag does not remove the content contained within the tag</p>	

For more information on the configuration settings for these refer to the appropriate sections in [Menu and Toolbar Item List](#)

## New User Interface Options

EditLive! 9 introduces a set of new configuration options and JavaScript APIs that allow you to create a minimalist editing UI for In-place editing. The new UI Configurations include the ability to

- float the main toolbar
- dock contextual toolbars
- remove the Preview, Design and Code view tabs
- minimise the toolbars by using the new megamenu collections described above
- remove the menus

In addition, you can use the new JavaScript APIs to

- close EditLive! when the end user clicks outside of the editable section
- vertically expand and contract content editable sections to better fit the content they contain.

For an example of using these new features refer to the quick start guide [Creating a Minimal Editing UI with In-place Editing](#).

## New Style Options

EditLive! 9 introduces the ability to both Blacklist or Whitelist the styles in the style sheet supplied to EditLive!. Whitelisting is where only those styles explicitly marked for inclusion in the stylesheet will appear in the Styles drop down. Blacklisting is where all styles, except those that are explicitly excluded will appear in the Styles drop down.

For more detailed explanation refer to the Developer Guide [Restricting what CSS classes appear in EditLive! styles drop down](#).

## Platform Support

EditLive! now requires Java 1.6 as the minimum version of Java.

Google Chrome on OS X 10.7 and above will now use Select Edit to display TinyMCE instead of EditLive!. This is because Chrome on OS X is a 32 bit application, however the Oracle Java plugin is a 64 bit application and will not function with Chrome as it is a 32 bit application.

For more information see the [System Requirements](#) section of this documentation.

## Behavioural Changes

### Loading Behaviour

EditLive! 9 features an updated loading screen that improves the user experience with more subtle load time visuals. The traditional loading dialogue with a progress meter has been replaced with a dimmed view of the Preview pane and spinner while EditLive! loads in the background. Once EditLive! has loaded the user immediately sees the editor interface.

## On-click Tab Switch

To enhance the user experience of EditLive!'s inline editing mode users no longer have to click on the Design tab to load the editing interface for an inline instance of EditLive!. In EditLive! 9 clicking within the Preview pane of an in-line instance of EditLive! will load the editing interface of EditLive!. The exception to this is if the user clicks on an interactive component within the preview - e.g. a Google Map, YouTube video etc - that interaction will not trigger a change from Preview to Design mode.

## Updated EditLive! UI

EditLive! 9 includes an updated user interface. While Tiny has worked to ensure the changes to existing UIs is minimal this update does affect some of the widths and heights of items in EditLive! and you are advised to test this with your configuration prior to updating your production environment. Changes in buttons widths in particular may cause your toolbar to wrap unexpectedly. The following lists some of the key changes:

- The introduction of chevrons to represent megamenus on the list, table and colour buttons increase the space these buttons require on the toolbar.
- The widths of the style, font and font size drop downs have been reduced. Long style or font names may now be truncated.
- The height of toolbars have been reduced.
- The spacing between toolbar button groups has been reduced.
- EditLive! now uses the Open Sans font by default. Changes to the font may affect the appearance of custom dialogs developed using the Advanced Java API.

## List Toolbar Buttons Converted to Megamenus

The Ordered and Unordered List toolbar buttons have been converted to megamenus - a convention used by most word processors today. These megamenus provide access to the different HTML list types available for each list. Previously changing the list type required the user to open the List Properties dialog.

## User-friendly Style Names

Class name that use naming conventions with camelCase, underscores or hyphens will be converted into user-friendly names. For more information refer to the *User Friendly Style Names* section of [Using CSS in EditLive!](#)

## Accessibility Checks

The W3C accessibility guidelines have been updated from WCAG 1.0 to WCAG 2.0. As per the standard, WCAG 1.0 "1" and "2" ratings have been replaced by the WCAG 2.0 "A" and "AA" ratings.

New accessibility as-you-type checks have been added for media elements including HTML5 media and embedded media.

The accessibility as-you-type check for absolute vs relative widths on DIVs and tables has been removed as it is no longer valid for WCAG 2.0.

## Media Preview

HTML5 and interactive embeddable media elements are now able to be previewed in the EditLive! Preview pane. They can also be interacted with in the Preview pane.

## Create and Remove Section Commands

Create Section (DIV) and Remove Section (DIV) have been enhanced to now provide the ability to insert and remove HTML 5 Semantic tags along with DIV and Blockquote elements. The configuration setting to add to the Toolbar and Menu remains the same as previous version.

## HTTP Clients

EditLive! 9.0 uses the Oracle (previously known as the "Sun") HTTP client layer to make requests by default. Most users will not notice any changes associated with this.

EditLive! 9.1 only uses the Oracle HTTP client layer. The "apache" layer has been removed.

## raiseEvent API Passes the Applet ID

raiseEvent applet callbacks are now passed the applet ID (the string used to create the EditLiveJava object) as the first function parameter. This enables developers to identify the editor instance that invoked the JavaScript call.

## WebDav Functionality Removed in 9.1

WebDav functionality has been removed, as of EditLive! 9.1. The following related configuration items no longer have any effect:

- [<webdav>](#)
- [<repository>](#)
- [<authentication>](#)
- [<realm>](#)

## JavaScript APIs Removed in 9.1

Any calls to this function will need to be removed:

- [setHttpLayerManager Method](#)

## Java APIs Removed in 9.1

See: [Java API](#)

# Quick Start Guide

This guide provides you with the information that you need to quickly get started developing with EditLive!

# Deploying EditLive! on a Web Server

EditLive! is a Java applet that integrates with web applications using a JavaScript API. When integrating EditLive! using the JavaScript API all of EditLive!'s code executes client side. The only server requirement is that the web server is able to host EditLive!'s source files. Each distribution of EditLive! is provided with a folder that can be deployed directly to your web server to use EditLive!.

When you download and unzip an EditLive! distribution it will contain a number of files and either:

- a *webfolder* directory, if you have downloaded the complete EditLive! SDK or
- an *editlivejava* directory, if you have downloaded an EditLive! update package

These directories are designed to be deployed to a web accessible location on a web server. Once you've done so you'll have access to EditLive! as part of your web application and you're ready to start integrating.

If you're using the complete SDK, the *webfolder* directory contains a collection of samples in several different scripting languages as well as the EditLive! runtime files. These examples can be accessed via the browser once deployed on your server. Within the SDK distribution the EditLive! runtime files are contained within the *webfolder/redistributables/editlivejava* folder.

With either distribution it is the *editlivejava* folder containing the EditLive! source files that you'll need to begin working with EditLive! within your own applications.



# Integrating EditLive! in 3 Lines

This article assumes that you've [deployed EditLive! to your web server](#) and you're ready to start integrating EditLive!.

The *editlivejava* directory now hosted on your web server includes all the files you need to get started with creating your own EditLive! integration.

This example takes you through the steps needed to put an instance of EditLive! into a web page.

The integration method used in this example is ideal when integrating EditLive! within a basic web form that requires rich text editing. It requires very little JavaScript and provides an active EditLive! instance in the location that the *show* method is called.

## 1. Start with a Basic HTML Page

### Basic HTML Page

```
<html>
  <head>
    <title>Simple 3 Line Integration</title>
  </head>
  <body>
    <h1>EditLive! will appear here</h1>
  </body>
</html>
```

## 2. Include the EditLive! JavaScript Library

The EditLive! JavaScript library includes all the code necessary to create an EditLive! instance. Even though EditLive! is a Java applet you never need to interact with Java directly, everything you need is handled by the JavaScript library.

In this case the location of the JavaScript library is given by *editlivejava/editlivejava.js*. You will need to adjust this URL depending on location of the EditLive! source files on your web server.

### Include the EditLive! JavaScript Library

```
<head>
  <title>Simple 4 Line Integration</title>
  <!-- Inline the EditLive! JavaScript Library -->
  <script src="editlivejava/editlivejava.js" language="JavaScript"></script>
</head>
```

## 3. Create, Configure and Show the EditLive! Instance

Now we will add the code to create an instance of EditLive!.

By default the instance of EditLive! will be configured using the sample configuration file contained in the *editlivejava* directory along with the editor source files.

The *show* method is called which creates the instance of EditLive! in the page based on the information passed in via the constructor and configuration file.

### Create, Configure and Show EditLive!

```
<h1>EditLive! will appear here</h1>
<!-- Create, Configure and Show the EditLive! Instance -->
<script type='text/javascript'>
  // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and a width of 700
  pixels.
  var editlive = new EditLiveJava("ELApplet", 700, 400);

  // .show is the final call and instructs the JavaScript library (editlivejava.js) to insert a new
  EditLive! instance
  // at the this location.
  editlive.show();
</script>
```

## Complete Code Example

### Create, Configure and Show EditLive!

```
<html>
  <head>
    <title>Simple 3 Line Integration</title>
    <!-- Inline the EditLive! JavaScript Library -->
    <script src="editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>
    <h1>EditLive! will appear here</h1>
    <!-- Create, Configure and Show the EditLive! Instance -->
    <script type='text/javascript'>
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and a width
      of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to insert a
      new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>
```

# Creating Editable Sections on a Page

The integration method used in this example is ideal when integrating EditLive! in a dynamically generated page, a page with existing content or when you require multiple editor instances within a page. It requires very little JavaScript and enables you to very easily create and manage multiple editor instances.

This article assumes that you've [deployed EditLive! to your web server](#) and you're ready to start integrating EditLive!.

The `editlivejava` directory now hosted on your web server includes all the files you need to get started with creating your own EditLive! integration.

This example takes you through the steps needed to put an instance of EditLive! into a web page by replacing existing DIV elements on the page.

## 1. Start with an Existing HTML Page that Contains DIVs

Start with a HTML page that includes DIVs. These elements will be replaced with editable sections in this example.

### HTML Page

```
<html>
<head>
<title>EditLive Demonstrations</title>
</head>
<body>
  <h1 style="text-align:center;">EditLive! Demonstrations</h1>
  <div style="width:700px;margin:auto">
    <h2 style=" text-align: left;">EditLive! - Works Like a Rich Text Editor Should</h2>
    <p><em>EditLive! is the rich text editor that works like a rich text editor should. No reformatting. No stress. No kidding.</em></p>
    <p><strong>Copy and paste</strong> content seamlessly, no matter where your content comes from: Microsoft Word, Excel or any web site.</p>
    <p><strong>Image editing</strong> is a breeze with EditLive!'s built in image editor.</p>
    <p><strong>Avoid mistakes. </strong> EditLive! includes out of the box capabilities such as <strong>spell check as you type</strong>, a<strong> thesaurus</strong> and<strong> hyperlink checking</strong>. And you get all of this without the need to license, install or configure any third party software.</p>
  </div>
  <p>&nbsp;</p>
  <div style="width:700px;margin:auto">
    <h2>Make Image Editing a Breeze</h2>
    <p>With a<strong> built-in image editor</strong>, EditLive! makes editing images a breeze.</p>
    <p>EditLive! is the only online rich text editor that enables you to:</p>
    <ul>
    <li>Resize and resample images</li>
    <li>Crop</li>
    <li>Rotate left and right</li>
    <li>Flip vertically and horizontally</li>
    <li>Add rounded corners, drop shadow and reflection effects</li>
    </ul>
  </div>
</body>
</html>
```

## 2. Include the EditLive! JavaScript Library

Include the EditLive! JavaScript library.

**Note:** You will need to adjust the URL for the script to map to the location of the `editlivejava` directory on your web server, based on your EditLive! deployment

### Include the EditLive! JavaScript Library

```
<script type="text/javascript" src="editlivejava/editlivejava.js"></script>
```

### 3. Add IDs to DIVs

Add IDs to the DIVs so that they can be addressed by the EditLive! JavaScript library. These IDs will be used to identify the DIVs that are to be replaced by EditLive!

#### Add IDs to DIVs

```
<div style='width:700px;margin:auto' id="FirstDIV">  
  
...  
  
<div style='width:700px;margin:auto' id="SecondDIV">
```

### 4. Create and Configure EditLive! Instances

Now construct an EditLive! instance and assign it to the DIVs.

#### Create and Configure EditLive! Instances

```
<script type="text/javascript">  
  //Create an EditLive! instance  
  editlive = new EditLiveJava("ELApplet");  
  //Assign the EditLive! instance to work with the named DIVs in the content  
  editlive.addEditableSection("FirstDIV");  
  editlive.addEditableSection("SecondDIV");  
</script>
```

## Complete Code

## HTML Page

```
<html>
<head>
<title>EditLive Demonstrations</title>
<script type="text/javascript" src="editlivejava/editlivejava.js"></script>
</head>
<body>
  <h1 style="text-align:center;">EditLive! Demonstrations</h1>
  <div style='width:700px;margin:auto' id="FirstDIV">
    <h2 style=" text-align: left;">EditLive! - Works Like a Rich Text Editor Should</h2>
    <p><em>EditLive! is the rich text editor that works like a rich text editor should. No
reformatting. No stress. No kidding.</em></p>
    <p><strong>Copy and paste</strong>&#160;content seamlessly, no matter where your content comes
from: Microsoft Word, Excel or any web site.</p>
    <p><strong>Image editing</strong> is a breeze with EditLive!'s built in image editor.</p>
    <p><strong>Avoid mistakes. </strong> EditLive! includes out of the box capabilities such as
<strong>spell check as you type</strong>, a&#160;<strong>thesaurus</strong>&#160;&and&#160;<strong>hyperlink
checking</strong>. And you get all of this without the need to license, install or configure any third party
software.</p>
  </div>
  <p>&nbsp;</p>
  <div style='width:700px;margin:auto' id="SecondDIV">
    <h2>Make Image Editing a Breeze</h2>
    <p>With a&#160;<strong>built-in image editor</strong>, EditLive! makes editing images a breeze.<
/p>
    <p>EditLive! is the only online rich text editor that enables you to:</p>
    <ul>
      <li>Resize and resample images</li>
      <li>Crop</li>
      <li>Rotate left and right</li>
      <li>Flip vertically and horizontally</li>
      <li>Add rounded corners, drop shadow and reflection effects</li>
    </ul>
  </div>
  <script type="text/javascript">
    //Create an EditLive! instance
    editlive = new EditLiveJava("ELApplet");
    //Assign the EditLive! instance to work with the named DIVs in the content
    editlive.addEditableSection("FirstDIV");
    editlive.addEditableSection("SecondDIV");
  </script>
</body>
</html>
```

# Creating a Minimal Editing UI with In-place Editing

In-place editing with EditLive! is ideal for when you want to create a minimalist editing UI. The code for the integration uses the same techniques as the [Creating Editable Sections example](#). It uses EditLive!'s configuration options and JavaScript API to minimize the visual appearance of the editor in the page. This enables you to construct interfaces that enable authors to quickly switch between reading content and editing content.

In-place editing is not a separate integration mode or methodology. Rather it is a visual design pattern that uses several EditLive! configuration options and JavaScript API settings to create a minimalist editing user interface. While this example uses several of these settings and APIs it's also possible to use these settings and APIs in isolation. For example, if you had an EditLive! instance where you wanted to remove the menu bar but have docked toolbars then this would be achieved by simply removing the menu bar as explained in step 3 of this guide - none of the other steps need be followed.

This article assumes that you've [deployed EditLive! to your web server](#) and you're ready to start integrating EditLive!.

The `editlivejava` directory now hosted on your web server includes all the files you need to get started with creating your own EditLive! integration.

This example takes you through the steps needed to put an instance of EditLive! into a web page using in-place editing. It is an extension of the [Creating Editable Sections example](#).

## 1. Integrate EditLive! Using Editable Sections

To begin with integrate EditLive! in the same way as the [Creating Editable Sections example](#).

Important



It is recommended that a doctype be used on your webpage to ensure that Internet Explorer does not use IE5 Quirks mode. Inline sections may not resize correctly in Internet Explorer if the page is using IE5 Quirks mode.

See the [W3C Recommended list of Doctype declarations](#) for information on doctype declarations that can be used.

## 2. Use a New Configuration File

Create a copy of the `sample_eljconfig.xml` file contained within the `editlivejava` directory and rename it as `inplace_editing.xml`. Point the EditLive! instance at this file using the `setConfigurationFile` method.

### Set the Configuration File

```
<script type="text/javascript">
  //Create an EditLive! instance
  var editlive = new EditLiveJava("ELApplet");
  //Assign the EditLive! instance to work with the named DIVs in the content
  editlive.setConfigurationFile("editlivejava/inplace_editing.xml");
  editlive.addEditableSection("FirstDIV");
  editlive.addEditableSection("SecondDIV");
</script>
```

## 3. Configure EditLive! to Use Floating Toolbars

In this step we're going to alter the configuration of EditLive! by editing the `inplace_editing.xml` file. We'll configure EditLive! to use a minimal set of floating toolbars to deliver rich text editing functionality. Each of the configuration options used in this step can be used independently and in isolation to alter the user interface of the editor. When used together though they enable you to create a streamlined, minimalistic editing interface without compromising on functionality.

### Remove the Menus, the Menu Bar and the About EditLive! menu

Find the `<menuBar>` element in your configuration file and remove the element and all its contents. This will remove menus from EditLive!'s interface.

Important



When you remove the About EditLive! menu by removing the `menuBar` element completely you must either use the `helpMenu` megamenu on the toolbar to make the About EditLive! dialog available or replicate the open source licensing information contained within the About EditLive! dialog in another place within your software and/or documentation.

### Use a Floating Toolbar

Set EditLive!'s main toolbar to appear as a floating toolbar within the editor by modifying the `<toolbars>` element.

### Float the Main Toolbar

```
<toolbars display="floating">
  ...
</toolbars>
```

## Minimize the Toolbar Button Set

To maximize the effectiveness of inline toolbars it's recommended that you use a minimal set of toolbar buttons. The following toolbar configuration item provides a toolbar appropriate for an EditLive! instance of approximately 700px wide.

This toolbar uses several megamenus to collapse commands under a single toolbar button.

### Note



Floating toolbars are still able to support multiple rows of toolbars and will wrap appropriately when the editor resizes. It's simply recommended to minimize the toolbar to improve the aesthetic of this integration method.

### Minimal Toolbar Configuration

```
<toolbars display="floating">
  <toolbar name="Command">
    <toolbarComboBox name="Style">
      <comboBoxItem name="P" />
      <comboBoxItem name="H1" />
      <comboBoxItem name="H2" />
      <comboBoxItem name="H3" />
      <comboBoxItem name="H4" />
      <comboBoxItem name="H5" />
      <comboBoxItem name="H6" />
    </toolbarComboBox>
    <toolbarSeparator />
    <toolbarButton name="textFormattingMenu" />
    <toolbarSeparator />
    <toolbarButton name="alignIndentMenu" />
    <toolbarSeparator />
    <toolbarButtonGroup name="List" />
    <toolbarSeparator />
    <toolbarButton name="HLink" />
    <toolbarButton name="ImageServer" />
    <toolbarButton name="InsertMedia" />
    <toolbarButton name="InsTableWizard" />
    <toolbarSeparator />
    <toolbarButton name="checkerMenu" />
    <toolbarSeparator />
    <toolbarButton name="AddComment" />
    <toolbarButton name="enableTrackChanges" />
    <toolbarSeparator />
    <toolbarButton name="helpMenu" />
    <toolbarButton name="pagewidth" />
    <toolbarButton name="Popout" />
  </toolbar>
</toolbars>
```

## Combine Inline Toolbars and Floating Toolbars

EditLive! provides a set of floating inline toolbars for table and image manipulation. To achieve a consistent visual effect these can be combined with the main toolbar as a dynamic toolbar by configuring the [<inlineToolbars>](#) element.

### Combine Floating Toolbars

```
<inlineToolbars showOnMainToolbar="true">
  ...
</inlineToolbars>
```

## 4. Turn Off EditLive!'s Preview, Design and Code Tabs

Removing EditLive!'s tabs and the borders associated with them will "hide" the editor. EditLive! will indicate an editable section in the document by displaying a surrounding blue border when the mouse hovers over the editable text. Tabs are removed by configuring the *tabPlacement* attribute of the `<wysiwygEditor>` element

### Turn Off EditLive!'s Tabs

```
<wysiwygEditor tabPlacement="off"></wysiwygEditor>
```

## 5. Use CloseOnFocusLost

Adding the `CloseOnFocusLost` method to your EditLive! objects will ensure that EditLive! closes when the user clicks outside of the editable section.

### Use CloseOnFocusLost

```
<script type="text/javascript">
  //Create an EditLive! instance
  var editlive = new EditLiveJava("ELApplet");
  //Assign the EditLive! instance to work with the named DIVs in the content
  editlive.setConfigurationFile("editlivejava/inplace_editing.xml");
  editlive.setCloseOnFocusLost(true);
  editlive.addEditableSection("FirstDIV");
  editlive.addEditableSection("SecondDIV");
</script>
```

## 6. Use ResizableSections

Adding the `ResizableSections` method to your EditLive! objects will enable them to vertically expand and contract to better fit the size of the content they contain. If using this setting in conjunction with the other options in this guide will result in EditLive!'s vertical scroll bar being removed. Instead the editor will expand down the page to fit its current content.

### Use ResizableSections

```
<script type="text/javascript">
  //Create an EditLive! instance
  var editlive = new EditLiveJava("ELApplet");
  //Assign the EditLive! instance to work with the named DIVs in the content
  editlive.setConfigurationFile("editlivejava/inplace_editing.xml");
  editlive.setCloseOnFocusLost(true);
  editlive.setResizableSections(true);
  editlive.addEditableSection("FirstDIV");
  editlive.addEditableSection("SecondDIV");
</script>
```

## Complete HTML and JavaScript Code



## Complete HTML and JavaScript Code

```
<!DOCTYPE HTML>
<html>
<head>
<title>EditLive Demonstrations</title>
<script type="text/javascript" src="editlivejava/editlivejava.js"></script>
</head>
<body>
  <h1 style="text-align:center;">EditLive! Demonstrations</h1>
  <div style='width:700px;margin:auto' id="FirstDIV">
    <h2 style="text-align:left;">EditLive! - Works Like a Rich Text Editor Should</h2>
    <p><em>EditLive! is the rich text editor that works like a rich text editor should. No
reformatting. No stress. No kidding.</em></p>
    <p><strong>Copy and paste</strong>&#160;content seamlessly, no matter where your content comes
from: Microsoft Word, Excel or any web site.</p>
    <p><strong>Image editing</strong> is a breeze with EditLive!'s built in image editor.</p>
    <p><strong>Avoid mistakes.</strong> EditLive! includes out of the box capabilities such as
<strong>spell check as you type</strong>, a&#160;<strong>thesaurus</strong>&#160;and&#160;<strong>hyperlink
checking</strong>. And you get all of this without the need to license, install or configure any third party
software.</p>
  </div>
  <p>&nbsp;</p>
  <div style='width:700px;margin:auto' id="SecondDIV">
    <h2>Make Image Editing a Breeze</h2>
    <p>With a&#160;<strong>built-in image editor</strong>, EditLive! makes editing images a breeze.<
/p>
    <p>EditLive! is the only online rich text editor that enables you to:</p>
    <ul>
    <li>Resize and resample images</li>
    <li>Crop</li>
    <li>Rotate left and right</li>
    <li>Flip vertically and horizontally</li>
    <li>Add rounded corners, drop shadow and reflection effects</li>
    </ul>
  </div>
  <script type="text/javascript">
    //Create an EditLive! instance
    var editlive = new EditLiveJava("ELApplet");
    //Assign the EditLive! instance to work with the named DIVs in the content
    editlive.setConfigurationFile("editlivejava/inplace_editing.xml");
    editlive.setCloseOnFocusLost(true);
    editlive.setResizableSections(true);
    editlive.addEditableSection("FirstDIV");
    editlive.addEditableSection("SecondDIV");
  </script>
</body>
</html>
```

## Complete Configuration

### Complete Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<editlive>
  <document>
    <html>
      <head>
        <link rel="stylesheet" href="yourserver.com/css/basic.css"/>
      </head>
      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
      -->
    <body>
```

```

        </body>
    </html>
</document>
<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
    <license
        domain="LOCALHOST"
        key="6FFF-7C38-9CFD-AE9B"
        licensee="For Evaluation Only"
        release="9.0"
        type="Evaluation License"
        productivityPack="true"
    />
</ephoxLicenses>
<spellCheck startBackgroundChecking="true" startAutoCorrect="true"/>
<htmlFilter
    outputXHTML="true"
    outputXML="false"
    xhtmlStrict="false"
    indentContent="false"
    logicalEmphasis="true"
    quoteMarks="false"
    uppercaseTags="false"
    uppercaseAttributes="false"
    wrapLength="0">
</htmlFilter>

    <!-- Set Tab Placement to "off" -->
<wysiwygEditor
    tabPlacement="off"
    brOnEnter="false"
    showDocumentNavigator="false"
    disableInlineImageResizing="false"
    disableInlineTableResizing="false"
    enableTrackChanges="false"
>
</wysiwygEditor>
<sourceEditor showBodyOnly="true"/>
<wordImport styleOption="clean"/>
<excelImport styleOption="merge_inline_styles"/>
<htmlImport styleOption="merge_inline_styles"/>
<plugins>
    <plugin name="autosave" />
    <plugin name="autolink" />
    <plugin name="insertHTML" />
    <plugin name="rtfpaste"/>
    <plugin name="setBackgroundMode"/>
    <plugin name="spelling" />
    <plugin name="tableToolbar" />
    <plugin name="accessibility" />
    <plugin name="imageEditor" />
    <plugin name="BrokenHyperlinkReport" />
    <plugin name="commenting" />
    <plugin name="templateBrowser" />
</plugins>

<templates>
</templates>
<accessibilityChecks
    errors="true"
    warnings="true"
    manual="true"
    WCAG1="true"
    WCAG2="true"
    Section508="true"
    inlineAccessibility="false"
    emptyImageAlt="error"
    tableMappingIssues="warn"
/>

```

```

<mediaSettings>
  <images allowLocalImages="true" allowUserSpecified="true" preferredWidth="800" preferredHeight="600">
    <imageDialog width="700" height="350" />
    <imageList>
    </imageList>
  </images>
  <multimedia>
    <services>
      <service name="YouTube" endpoint="http://www.youtube.com/oembed" scheme="http://*.youtube.com
/*" />
      <service name="Twitter" endpoint="https://api.twitter.com/1.1/statuses/oembed.json" scheme="
*twitter.com/*" />
      <service name="Vimeo" endpoint="http://vimeo.com/api/oembed.json" scheme="http://vimeo.com/*" />
      <service name="Slideshare" endpoint="http://www.slideshare.net/api/oembed/2" scheme="http://www.
slideshare.net/*/*" />
      <service name="Flickr" endpoint="http://www.flickr.com/services/oembed/" scheme="http://*.
flickr.com/*" />
      <service name="Instagram" endpoint="http://api.instagram.com/oembed" scheme="http://www.
instagram.com/*/*" />
      <service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed" scheme="*" />
    </services>
    <types>
    </types>
  </multimedia>
</mediaSettings>
<hyperlinks>
  <hyperlinkList>
    <hyperlink href="http://docs.ephox.com/display/EditLive/hyperlinkList" description="How To Update
This List" />
    <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
    <hyperlink href="http://docs.ephox.com" description="Ephox Documentation" />
    <hyperlink href="http://support.ephox.com/" description="Ephox Support" />
    <hyperlink href="http://releases.ephox.com" description="Ephox Releases" />
  </hyperlinkList>
  <mailtoList>
    <mailtoLink href="mailto:sales@ephox.com" description="Ephox Sales" />
  </mailtoList>
</hyperlinks>

  <!-- Remove the menuBar element -->

  <!-- Minimize the toolbar configuration and float the toolbar-->
<toolbars display="floating">
  <toolbar name="Command">
    <toolbarComboBox name="Style">
      <comboBoxItem name="P" />
      <comboBoxItem name="H1" />
      <comboBoxItem name="H2" />
      <comboBoxItem name="H3" />
      <comboBoxItem name="H4" />
      <comboBoxItem name="H5" />
      <comboBoxItem name="H6" />
    </toolbarComboBox>
    <toolbarSeparator />
    <toolbarButton name="textFormattingMenu" />
    <toolbarSeparator />
    <toolbarButton name="alignIndentMenu" />
    <toolbarSeparator />
    <toolbarButtonGroup name="List" />
    <toolbarSeparator />
    <toolbarButton name="HLink" />
    <toolbarButton name="ImageServer" />
    <toolbarButton name="InsertMedia" />
    <toolbarButton name="InsTableWizard" />
    <toolbarSeparator />
    <toolbarButton name="checkerMenu" />
    <toolbarSeparator />
    <toolbarButton name="AddComment" />
    <toolbarButton name="enableTrackChanges" />
    <toolbarSeparator />
    <toolbarButton name="helpMenu" />
  </toolbar>

```

```

        <toolbarButton name="Popout" />
    </toolbar>
</toolbars>
    <!-- Display the inline toolbars contextually on the main toolbar -->
<inlineToolbars showOnMainToolbar="true">
    <inlineToolbar name="img">
        <toolbarButton name="rotateCCW" />
        <toolbarButton name="rotateCW" />
        <toolbarSeparator />
        <toolbarButton name="flipVertical" />
        <toolbarButton name="flipHorizontal" />
        <toolbarSeparator />
        <toolbarButton name="reflect" />
        <toolbarButton name="dropShadow" />
        <toolbarButton name="roundedCorners" />
        <toolbarSeparator />
        <toolbarButton name="crop" />
    </inlineToolbar>
    <inlineToolbar name="table">
        <toolbarButton name="InsRow" />
        <toolbarButton name="InsCol" />
        <toolbarButton name="DelRow" />
        <toolbarButton name="DelCol" />
        <toolbarButton name="DelTable" />
        <toolbarSeparator />
        <toolbarButton name="Split" />
        <toolbarButton name="Merge" />
        <toolbarSeparator />
        <toolbarButton name="tableautofit" />
        <toolbarButton name="percentageTableSizing" />
        <toolbarButton name="pixelTableSizing" />
        <!-- Enterprise Edition Features -->
        <toolbarSeparator />
        <toolbarButton name="ApplyCellHeaders" />
        <toolbarButton name="ClearCellHeaders" />
        <toolbarButton name="TableHeaderMappings" />
        <toolbarSeparator />
        <toolbarButton name="Gridlines" />
    </inlineToolbar>
</inlineToolbars>
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo" />
        <shrtMenuItem name="Redo" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Cut" />
        <shrtMenuItem name="Copy" />
        <shrtMenuItem name="Paste" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Select" />
        <shrtMenuSeparator />
        <!-- Enterprise Edition Features -->
        <shrtMenuItem name="AddComment" />
        <shrtMenuItem name="acceptChange" />
        <shrtMenuItem name="rejectChange" />
        <shrtMenuItem name="nextchange" />
        <shrtMenuItem name="previouschange" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Hyperlink" />
        <shrtMenuItem name="RemoveHyperlink" />
        <shrtMenuItem name="PropImage" />
        <shrtMenuItem name="EditMedia" />
        <shrtMenuItem name="PropObject" />
        <shrtMenuItem name="PropList" />
        <shrtMenuItem name="PropHR" />
        <shrtMenuItem name="PropSection" />
        <shrtMenuItem name="PropForm" />
        <shrtMenuItem name="PropFormField" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Split" />
        <shrtMenuItem name="Merge" />
    </shrtMenu>
</shortcutMenu>

```

```
<shrtMenuItem name="tableautofit"/>
<shrtMenuSeparator/>
<shrtMenuItem name="PropTable"/>
<shrtMenuItem name="PropRow"/>
<shrtMenuItem name="PropCol"/>
<shrtMenuItem name="PropCell"/>
<shrtMenuSeparator/>
<shrtMenuItem name="synonyms"/>
<shrtMenuItem name="EditTag"/>
</shrtMenu>
</shortcutMenu>
</editlive>
```

# Install Guide

Tiny EditLive! empowers business users with an easy-to-use, intuitive interface for creating and publishing web content.

EditLive!'s open standards-based platform enables developers to build, deploy, and run powerful content management applications.

## Highlights of EditLive!:

- Empowers non-technical business users to create and publish web content with an easy-to-use familiar interface.
- Separates content structure from style to ensure site style and design integrity
- The ability to deploy content to multiple sites or devices
- Deployed and upgraded instantly over the Internet or intranets.
- XML and XHTML open standards support.
- Cross-platform Windows, Macintosh OS X, and Linux clients.
- Supports a highly scalable n-tier architecture.
- Open platform for J2EE and Microsoft IIS servers enables developers to control, customize, and extend.


## Contents:

- [System Requirements](#)
- [General Server Install Instructions](#)
- [Installation Guide - JavaScript](#)
- [Installation Guide - IIS Install](#)
- [Installation Guide - ASP.NET](#)
- [Deploying to an External Web Server](#)
- [Installing TinyMCE](#)
- [Client Install](#)
- [Applet Security - Deployment Ruleset](#)
- [Java SWT Framework](#)

# System Requirements

## Client Requirements


### Java Version Support

 Java 8 is supported in EditLive 9.1 and later. We recommend using the latest release of Java 8. Java 8 is *not* supported on EditLive 8 and 9.0. For these releases, we recommend using the latest release of Java 7.

If using Java 6, we recommend using the latest public release (6u45), or upgrading to Java 7 or 8.  
If using Java 7, we recommend using the latest public release (7u80). A minimum of Java 7u4 is required.

EditLive requires Oracle's distribution of Java. Other runtimes such as OpenJDK are not supported.  
Google Chrome Support

 Google has discontinued support for Java in Chrome. See [this article](#) for more details.  
Firefox for Windows 64-bit Version Support

 Firefox for Windows started publishing 64-bit builds as of Firefox 43. These 64-bit builds do not support Java. Please use 32-bit Firefox for Windows.

Note that Firefox 64-bit for Linux does support Java.

### Microsoft Windows 8.1, 8, 7, Vista, XP, Server 2008:

- Oracle Java Runtime Environment 6, 7 or 8
- Internet Explorer 7+ (IE 11 recommended)
- Firefox 22-51 (Firefox 52+ no longer supports Java) (32-bit version only)

### Apple Mac OS X 10.7+:

- Oracle Java Runtime Environment 7 or 8
- Safari 6+
- Firefox 22-51 (Firefox 52+ no longer supports Java)

### Apple Mac OSX 10.6:

- Java Runtime Environment 6 update 5 or later (supplied by Apple)
- Safari 5.1+
- Firefox 22-51 (Firefox 52+ no longer supports Java)

### Linux:

- Oracle Java Runtime Environment 6, 7, 8
- Firefox 22-51 (Firefox 52+ no longer supports Java)
- Ubuntu or Linux Mint recommended

### EditLive! Microsoft Word Import Feature:

- Microsoft Windows 8.1, 8, 7, Vista, XP
- Microsoft Office 2000 or greater installed on the client machine

## Supported Application Servers

The EditLive! distribution can be integrated using JavaScript on any web server that HTML content can be served from.

# General Server Install Instructions

The EditLive! SDK is packaged with examples, image upload scripts, and the EditLive! application files allowing EditLive! to be run and redistributed for other applications. The examples for EditLive! are packaged within the *webfolder* folder which must be deployed on a Web server to allow the integrations to be viewed.

EditLive! comes packaged as a .zip file or an installer package depending on the EditLive! SDK to be used. To install EditLive!, either unzip the package to a specified directory or run the installer. Once EditLive! has been installed, the Web components of the install can either be found within a folder named *webfolder*, which is a subfolder of the root folder, or within a Java Web archive (.war file). Placing this Web archive or folder on a Web server will allow these portions of the SDK to be accessed.

## Deploying the EditLive! Examples

The examples for EditLive! are packaged within the folder named *webfolder*. This folder should be placed in a Web accessible directory in order to access the EditLive! sample integrations.

When placing the *webfolder* directory on a Web server it must be ensured that its child directories are also Web accessible.

Once the *webfolder* directory has been placed on a Web server the EditLive! tutorials and examples can be accessed using a Web browser. A listing of the EditLive! for Java examples is available by accessing the URL *http://YOURSEVER/WEBFOLDER/* where *YOURSEVER* represents the host name of the Web server EditLive! has been installed on and *WEBFOLDER* is the name of the virtual directory mapping to the EditLive! *webfolder* directory on the file system.

## EditLive! Evaluation License

EditLive! is packaged with a license for the *localhost* host. This is an untimed license that allows EditLive! to be accessed via the *localhost* domain. Developers can use this license while evaluating and developing with EditLive!. When deploying EditLive! within a Web application, the license for the *localhost* domain should be replaced with a license for the relevant Web application host.

For more information on the terms of EditLive! licenses, please see the *license.txt* file packaged with EditLive!. This file can be found at *SDK\_INSTALL/license.txt*, where *SDK\_INSTALL* is the location that the EditLive! SDK has been installed to.

## Redistributing EditLive!

The files required to redistribute EditLive! can be found in the *SDK\_INSTALL/webfolder/redistributables* directory, where *SDK\_INSTALL* represents the location that the SDK has been installed to. The *redistributables* directory contains all the files required to deploy EditLive! to a Web server for use within a Web application.

To deploy EditLive! with another Web application, copy the *redistributables* directory, along with its subdirectories, into the relevant Web application and ensure the following values are correct for the Web application:

- URL for the [setDownloadDirectory Method](#)
- Ensure you've included the EditLive! Javascript file

For more information, see the EditLive! [Tutorials](#) and [Examples](#).

The **redistributables** directory can be renamed, as can its subdirectories, without affecting the functionality of EditLive! provided that the URLs for the relevant properties, as listed above, are changed in accordance with the changes made to the structure, location, and name of the *redistributables* directory and its contents.

## Packaged Image Upload Handler Scripts

Packaged with the EditLive! SDK are a range of image upload handler scripts which can be used to receive images uploaded by EditLive! via HTTP. The packaged image upload scripts can be found in the *SDK\_INSTALL/webfolder/uploadscripts/directory*.

For more information on how to use EditLive!'s integrated image upload functionality and the packaged image upload scripts, please see the documentation on [HTTP Upload Support for Images and Objects](#).

## Packaged End User Help

Localized basic end user help files are also distributed with EditLive!. These can be found in the *editlivejava-enduserhelp.zip* zip file (which can be unzipped using an un-archiving program such as [WinZip](#) or [Power Archiver](#)) and can be redistributed with EditLive!. End user help is currently available in the following languages:

- English
- Arabic
- Catalan
- Chinese (Traditional)
- Chinese (Simplified)
- Croatian
- Czech
- Danish
- Dutch
- Farsi
- Finnish



- French
- German
- Greek
- Hebrew
- Hungarian
- Italian
- Japanese
- Korean
- Norwegian
- Polish
- Portuguese
- Portuguese (Brazilian)
- Romanian
- Russian
- Slovak
- Slovenian
- Spanish
- Swedish
- Thai
- Turkish
- Ukrainian

Each version of the end user help is contained within its own directory and can therefore be redistributed separately to other versions of the end user help.

## See Also

- [Minimizing an EditLive! Deployment](#)
- [HTTP Upload Support for Images and Objects](#)
- [Licensing EditLive!](#)

# Installation Guide - JavaScript

This document outlines the steps needed to install the Web component of the Tiny EditLive! development suite.

## Installation Details

The Tiny EditLive! SDK will be provided as a .zip file. The first step you need to undertake is to unzip the downloaded file. In order to do this, simply use an unzipping program such as [WinZip](#) or [Power Archiver](#). You should unzip the file to an appropriate location, such as C:\editlive\ (on a Windows machine). Once you have done this, you need to set up Tiny EditLive! to run on your Web server as per the instructions below.

### Web Server Deployment

To deploy the EditLive! SDK on your Web server, simply copy the *editlive* directory and all its subdirectories from the location where they were un-archived to a location from which they can be served on your Web server.

## What to do When You Have Finished the Installation Process

If you have followed this installation guide correctly, you can now start using EditLive!. Please work through the Tiny EditLive! documentation and tutorials on this site as well as the packaged examples to familiarize yourself with the product.

# Installation Guide - IIS Install

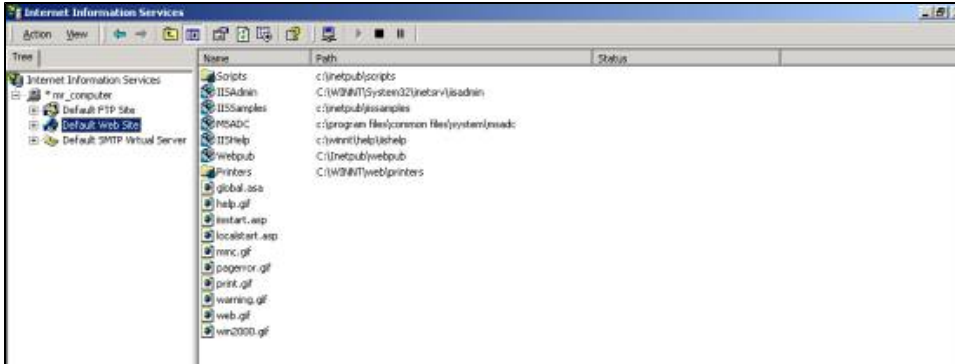
This document outlines the steps needed to install the Web component of the Tiny EditLive! development suite.

## Installation Details

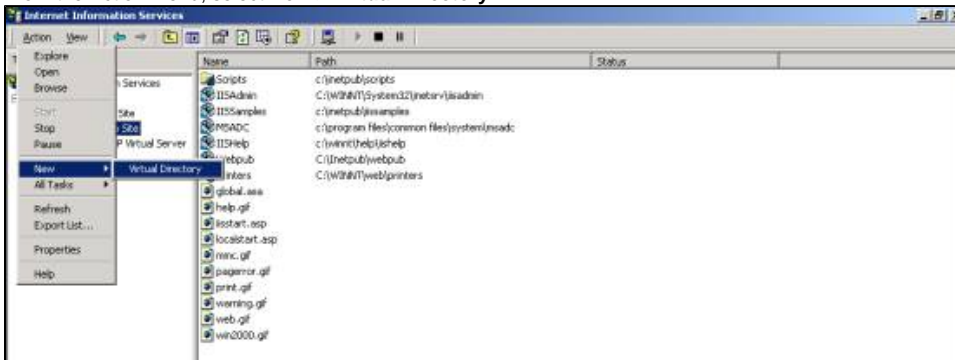
After installing EditLive! on a Windows machine running IIS (by unzipping the *editlive.zip* file), the next step is to map a virtual directory for EditLive!.

To map a virtual directory you will need to follow the steps below:

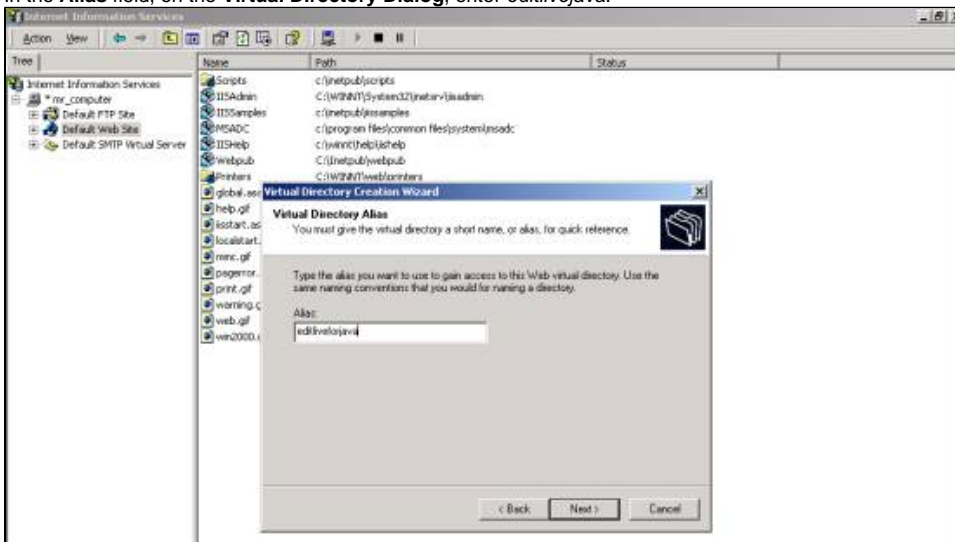
1. On the **Start** menu, in **Settings**, go to **Control Panel**. Double-click on **Administrative Tools** and then **Internet Services Manager**.
2. Expand the tree next to your Computer Name.
3. Select **Default Web Site**.



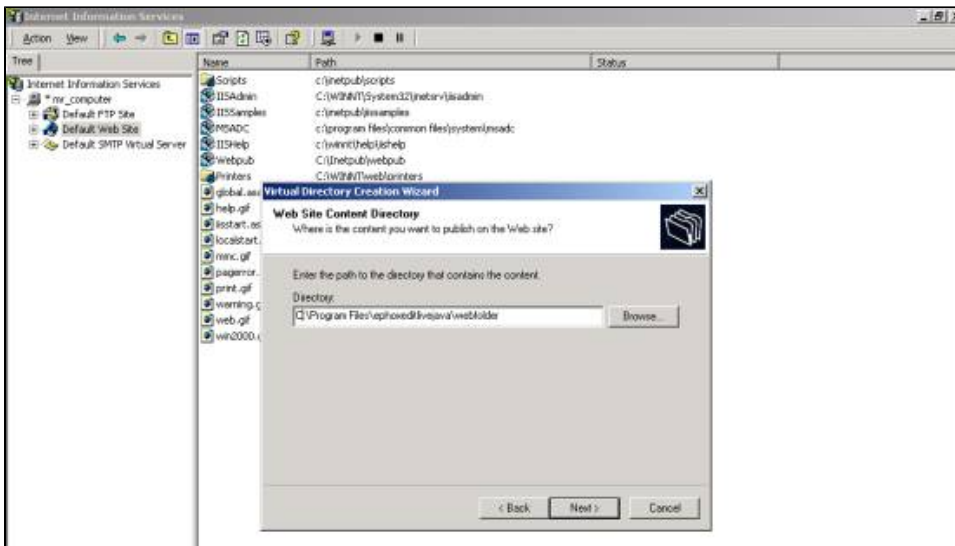
4. From the Action menu, select **New > Virtual Directory**.



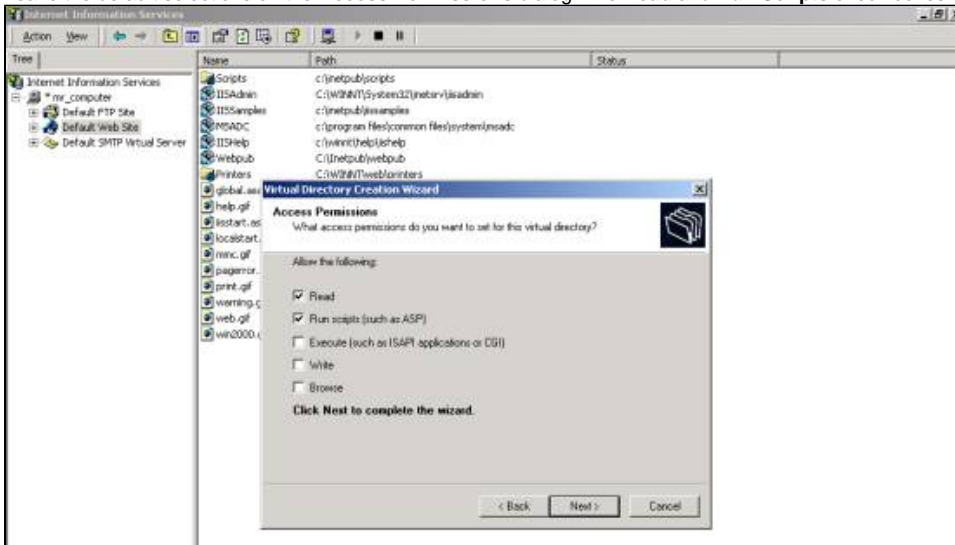
5. Follow the Virtual Directory Creation Wizard.
6. In the **Alias** field, on the **Virtual Directory Dialog**, enter *editlivejava*.



7. In the Directory field, on the **Virtual Site Content Directory Dialog**, enter the location *EDITLIVEINSTALL\webfolder*, where *EDITLIVEINSTALL* is the location where EditLive! is installed (e.g. "*C:\Program Files\Ephox EditLive!\webfolder*").



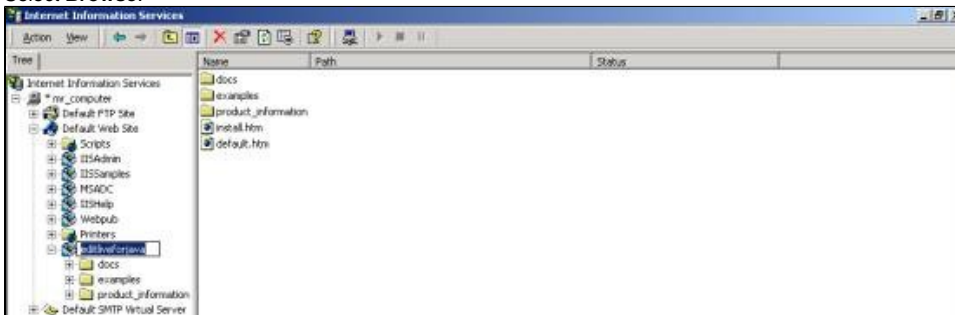
8. Leave the default selections on the **Access Permissions** dialog. The **Read** and **Run Scripts** check boxes should be selected.



9. Click **Next**, followed by **Finish**, to exit the wizard.

To check the installation process has been completed correctly:

1. Open a Web browser.
2. In the IIS snap-in, in the webfolder directory you just created, right-click on *default.htm*.
3. Select **Browse**.



You should see the Tiny EditLive! Web component home page within the Web browser. If so, you have successfully installed EditLive! and customized your system correctly.

## Once You Have Successfully Installed Tiny EditLive!

You can now start using Tiny EditLive!.

If you are using a Web server on your machine to serve the EditLive! SDK then you may use a Web browser to see examples. Direct your browser to: <http://localhost/editliveWebfolder> where *editliveWebfolder* is the name of the virtual directory in IIS.

# Installation Guide - ASP.NET

## Introduction

The Tiny EditLive! ASP.NET Server Control has been designed to allow easy integration of the WYSIWYG HTML editing capabilities of EditLive! into ASP.NET Web Forms. The EditLive! ASP.NET Server Control has been designed to allow for seamless interaction between the ASP.NET architecture and the EditLive! applet. Once the required files have been included in an ASP.NET Web project EditLive! can be easily included in any of your ASP.NET Web Forms.

## Requirements

- IIS 6.0 or 7.0 (for deployment)
- .NET framework 3.0, 3.5 or 4.0
- Visual Studio 2005 or later (for development)
- EditLive! 8.1 or above.

## Installing the EditLive! ASP.NET Server Control

When using the EditLive! ASP.NET Server Control, there are several steps that must be followed. The server control file, *EditLiveJavaControl.dll*, must be installed in the relevant project. In addition to this, the EditLive! source files and libraries must be deployed on the same Web server as the relevant project. Finally, to provide IntelliSense (popup context menu) support for the EditLive! Server Control in the HTML view of the Web Forms editor, an XML schema file must be placed in a specific directory.

### Installing the EditLive! Server Control with the Toolbox

The EditLive! Server Control may be added to the Microsoft Visual Studio .NET Toolbox to allow developers to create an instance of the EditLive! Server Control on a page via the drag-and-drop mechanism. To use the EditLive! Server Control in this manner the following steps must be followed:

1. In the Microsoft Visual Studio IDE, select the **Tools > Customize Toolbox...** menu item.
2. Select the **.NET Framework Components** tab of the **Customize Toolbox** dialog.
3. Click **Browse...**
4. Browse to the location of the *EditLiveJavaControl.dll* file. This file can be found in the *EditLiveControl\_INSTALL/webfolder/aspnet/servercontrol/* directory, where *EditLiveControl\_INSTALL* represents the location you installed the control to.
5. With the *EditLiveJavaControl.dll* file selected, click **Open**.
6. Ensure that the EditLiveJava control is now available in the list of .NET Framework Components and that the box next to the name of the control is checked.
7. Click **OK**.
8. The EditLive! Server Control should now be available as part of the Microsoft Visual Studio .NET Toolbox.

When using the EditLive! Server Control via the Microsoft Visual Studio .NET Toolbox, the relevant reference will automatically be added to the project when the first instance of the EditLive! Server Control is created.

### Installing the EditLive! Server Control as a Project Reference

If the EditLive! Server Control cannot be made available through the use of the Microsoft Visual Studio .NET Toolbox, or you do not wish to make it available, then a reference to it can be added directly to the project. To use the EditLive! Server Control in this manner the following steps must be followed:

1. In the Microsoft Visual Studio .NET IDE, select the **Project > Add Reference...** menu item.
2. Select the **.NET tab** of the **Add Reference** dialog.
3. Click **Browse...**
4. Browse to the location of the *EditLiveJavaControl.dll* file. This file can be found in the *EditLiveControl\_INSTALL/webfolder/aspnet/servercontrol/* directory, where *EditLiveControl\_INSTALL* represents the location you installed the SDK to.
5. With the *EditLiveJavaControl.dll* file selected, click **Open**.
6. Ensure that the *EditLiveJavaControl.dll* file appears in the **Selected Components** list.
7. Click **OK**.

The EditLive! Server Control is now available to reference from the project. Instances of the control can now be created using the tag associated with it.

## Other Files Required to Use EditLive! in an ASP.NET Web Application

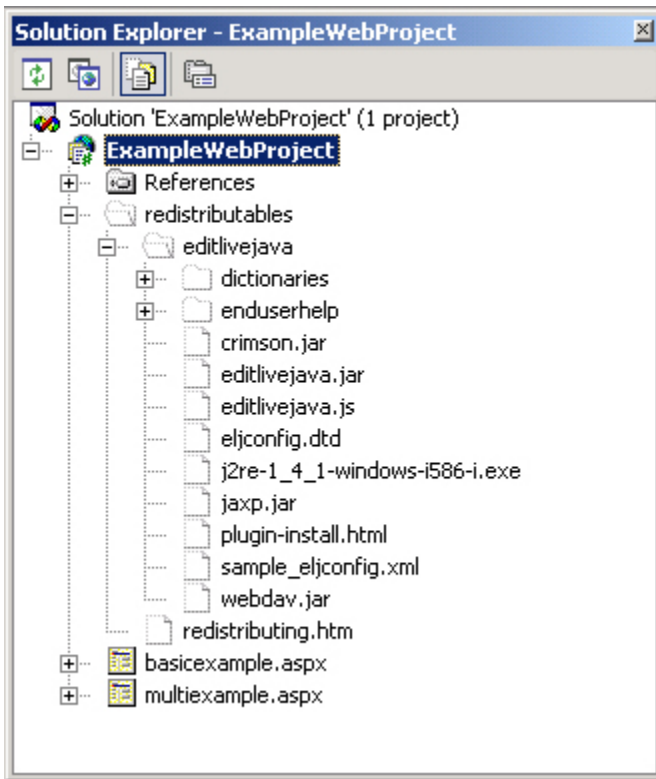
In addition to including the *EditLiveJavaControl.dll* file in your ASP.NET project, either through the ASP.NET Toolbox or a project reference, the EditLive! source files and libraries must be present on the same Web server as the relevant project. It is recommended that, in order to ensure that these files are present on the same Web server as the relevant project, they be added to the project itself. These files can be found within the *redistributables/editlivejava* directory and its subdirectories.

- The *redistributables/editlivejava* directory contains the source files and libraries for use with EditLive!. The files in this directory must be present on the Web server in order to instantiate EditLive!. This directory must be available on the same Web server as the relevant project as relative links to it must be able to be created.

As the Microsoft Visual Studio .NET IDE does not allow directories and their contents to be recursively added to a project, the following method describes a way to easily add the EditLive! source files and libraries to your .NET Web Project:

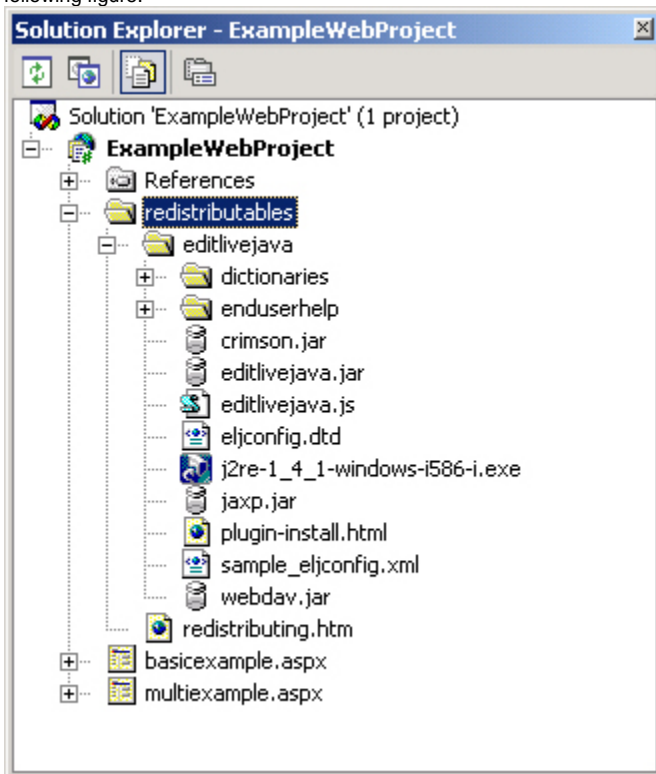
1. Copy the *redistributables* directory and all its subdirectories, which contain the EditLive! for Java source files and libraries, to the relevant Web project directory so that the *redistributables* directory is a sub-directory within the Web project.
2. Open the Solution Explorer window by selecting the **View > Solution Explorer** menu item.
3. Once the Solution Explorer window is open select the **Project > Show All Files** menu item to reveal all the files within the Web project's directories.

If the **Show All Files** option is already selected before performing this step it may have to be unselected and then reselected.



4. In the Solution Explorer, select the root directory for the EditLive! source files and libraries which have not yet been included in the project (files which are not included are represented as uncolored icons, as above). In the example above the relevant directory is the *redistributables* directory.
5. With the relevant root directory selected (in the above example, the *redistributables* directory), select the **Project > Include In Project** menu item.

6. The Solution Explorer window should now have colored representations of the EditLive! source files and libraries. It should look similar to the following figure.



## Constraints

By default ASP.NET forms cannot post values containing HTML. In order to successfully post EditLive!'s content, you will need to ensure the page directive contains the **validateRequest="false"** parameter.

In ASP.NET 4.0, you also need to add the following to your web.config, in the <system.web> section:

```
<httpRuntime requestValidationMode="2.0" />
```



# Deploying to an External Web Server

This document outlines how to deploy the Ephox EditLive! SDK to a Web server external to your local machine.

In order to deploy to an external Web server you must have write permissions for the directory on the Web server where you wish to deploy the EditLive! SDK. Ensure that you have the correct SDK for your Web server architecture.

## Deployment Details

To deploy the EditLive! SDK to an external Web server:

1. Install EditLive! on your local machine.
2. Locate the directory where you installed EditLive!.
3. Copy the *webfolder* subdirectory to the location where you wish to deploy it to on the external Web server.
4. To access the EditLive! SDK on the external Web server direct your browser to *http://YOUR\_WEB\_SERVER/LOCATION\_OF\_EDITLIVE\_WEBFOLDER/*
  - a. *YOUR\_WEB\_SERVER* - name of the external web server you have deployed to
  - b. *LOCATION\_OF\_EDITLIVE\_WEBFOLDER* - the location to which the EditLive! SDK webfolder directory was copied to in the above steps

The EditLive! SDK should now be ready to use from the external Web server.

If you are using the EditLive! J2EE SDK then please consult the documentation for your server for information on how to achieve deployment of a Web archive (.war file) to a remote server.

# Installing TinyMCE

**Select Edit works with TinyMCE 3.x only.**

TinyMCE is a JavaScript rich text editor supported by Tiny.

## Deploying TinyMCE

You can obtain an Tiny-supported build of TinyMCE from the [TinyMCE website](#).

You can integrate TinyMCE into your application using either the official TinyMCE APIs (documented with the TinyMCE distribution) or the Tiny [Select Edit APIs](#). The Tiny [Select Edit APIs](#) are a subset of the EditLive! APIs, meaning developers can design their applications for use with both EditLive! and TinyMCE while only dealing with a single API library.

To install TinyMCE for use with the [Select Edit APIs](#), please perform the following steps:

1. Copy the zip file obtained from the [TinyMCE website](#).
2. Locate where you have installed EditLive! for use by your application (i.e. where the *redistributables/editlivejava* directory and its contents are located). For more information, consult the [General Server Install Instructions](#).
3. Paste the TinyMCE zip archive into the *redistributables/editlivejava/expressEdit* directory.
4. Unzip the archive into this directory.

For more information on the Tiny Select Edit APIs, please see the EditLive! [Developer Guide](#).

See Also

- [Using TinyMCE](#)
- [setExpressEdit Method](#)
- [General Server Install Instructions](#)

# Client Install

The EditLive! applet is automatically deployed to users through Java's applet deployment technology. This allows users to have EditLive! seamlessly installed when first running a page containing EditLive!. Users are not required to physically download and install EditLive! themselves. This greatly simplifies the process for the end user.

## What is Required to Use EditLive!

EditLive! requires that the Java Runtime Environment (JRE) version 1.6 or above is installed on the user's computer. If this is not detected then EditLive! will automatically deploy and install a version of the JRE.

## Getting the Java Runtime Environment

Copies of Sun Microsystem's Java Runtime Environment can be found on the [Java Web site](#). Developers can download JRE installers from the Java Web site for hosting on their servers for the purposes of deploying the JRE.

## Installing the EditLive! Client

EditLive! is automatically deployed to users via the Java Applet technology using signed JAR files. All the files necessary to achieve the deployment of EditLive! are included in the SDK and can easily be hosted on a Web server.

When a page containing EditLive! is first accessed by a user, the client machine downloads the files necessary to run EditLive! and permanently caches them on the client machine. When accessing the page in future, the files necessary to run EditLive! will be loaded from the cache.

If the distribution of EditLive! is updated on the server, any updated files are automatically deployed to the client upon their first access and cached.

# Applet Security - Deployment Ruleset

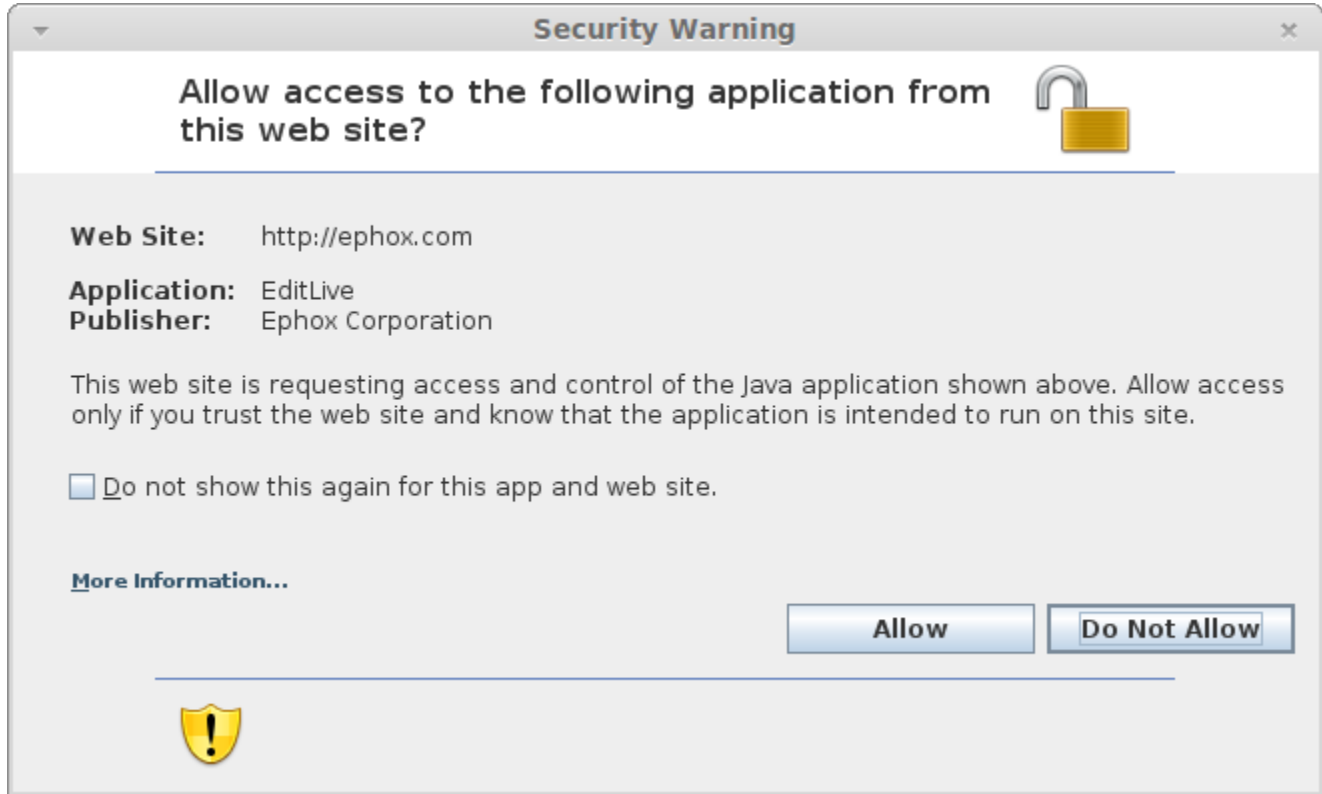
Note:

This tutorial contains information from the following sources:  
[http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment\\_rules.html](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment_rules.html)  
[https://blogs.oracle.com/java-platform-group/entry/deployment\\_rule\\_set\\_by\\_example](https://blogs.oracle.com/java-platform-group/entry/deployment_rule_set_by_example)

Please refer to these resources for further information regarding Deployment Rule Sets and their elements.

Recent changes to the Java web browser plugin security model may result in users receiving security prompts when they attempt to run Java applets in their web browser.

An example of this prompt can be seen below:



Clicking the "Do not show this again for this app and web site." checkbox should prevent this dialog from appearing in future. However, System Administrators may find it desirable for these prompts to not appear at all.

These dialogs can be avoided by creating and implementing a **Deployment Rule Set** (available since JRE 7 update 40.)

## Deployment Rule Set Creation

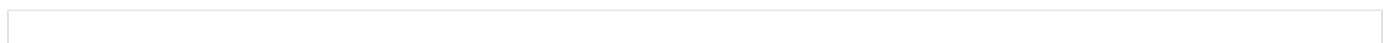
Note:

This section focuses specifically on creating a rule that matches a location. A location matching rule is required to permit EditLive to access its Javascript APIs.

For alternate rule configurations, please refer to the following documentation:  
[http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment\\_rules.html#define](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment_rules.html#define)

The Rule Set is defined an XML file that must be named "*ruleset.xml*".

The contents of this XML file should look similar to the following example:



```
<ruleset version="1.0+">
  <rule>
    <id location="http://YOUR_DOMAIN_HERE" /> <!-- For example: <id location="http://*.ephox.com" /> -->
    <action permission="run" />
  </rule>

  <rule>
    <id>
      <certificate algorithm="SHA-256" hash="794F53C746E2AA77D84B843BE942CAB4309F258FD946D62A6C4CCEAB8E1DB2C6"
/> <!-- Oracle's public certificate hash. Having this will allow things like the Java.com secure version check
applet. -->
    </id>
    <action permission="run" />
  </rule>
</ruleset>
```

This sample Rule Set contains 2 rules:

1. Allow applications from the location "http://YOUR\_DOMAIN\_HERE" to run. This address should be modified to match the location that EditLive! is being accessed from. For example: "\*.ephox.com".
2. Allow applications signed with the Oracle public certificate to run.

Additional rules can be defined as required. For information regarding the location definition, please refer to the following:

[http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment\\_rules.html#define](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment_rules.html#define)

This example can be downloaded here: [sample.zip](#)

## Create + Sign Deployment Rule Set JAR

Once the Rule Set has been created, the **ruleset.xml** file must be packed into a signed JAR file for deployment.

To create the JAR, open a shell and run the following commands:

```
cd <ruleset.xml_directory>
jar -cf DeploymentRuleSet.jar ruleset.xml
```

The exact procedure for signing the JAR will vary between users. Please refer to the following documentation for more information regarding creating and signing JAR files:

<http://docs.oracle.com/javase/tutorial/deployment/jar/index.html>

## Installing the Deployment Rule Set JAR

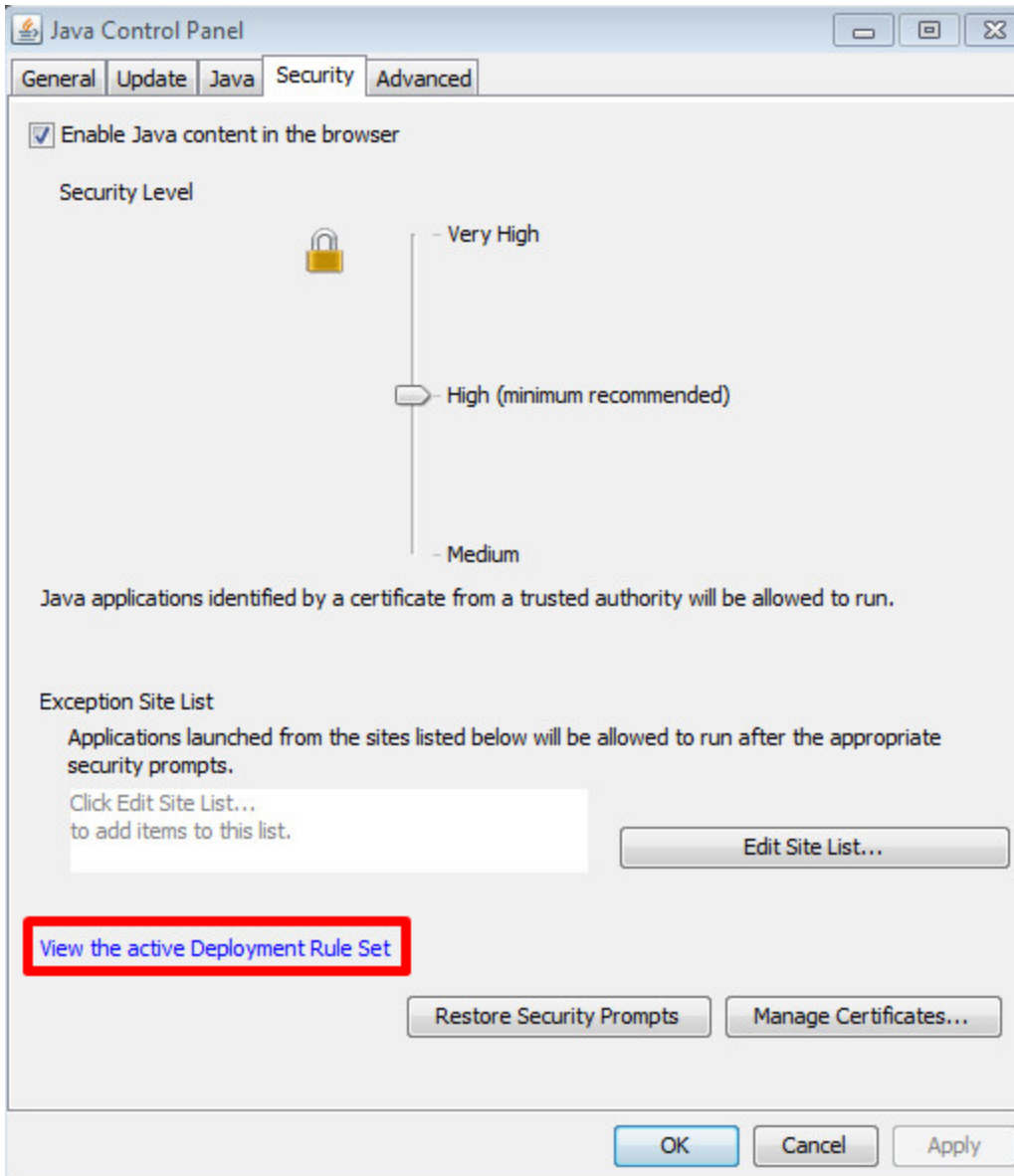
The signed JAR must be deployed to the client's environment for the Rule Set to take effect. The installation location is operating system dependent.

For information regarding the installation location of the Rule Set JAR, please refer to the following:

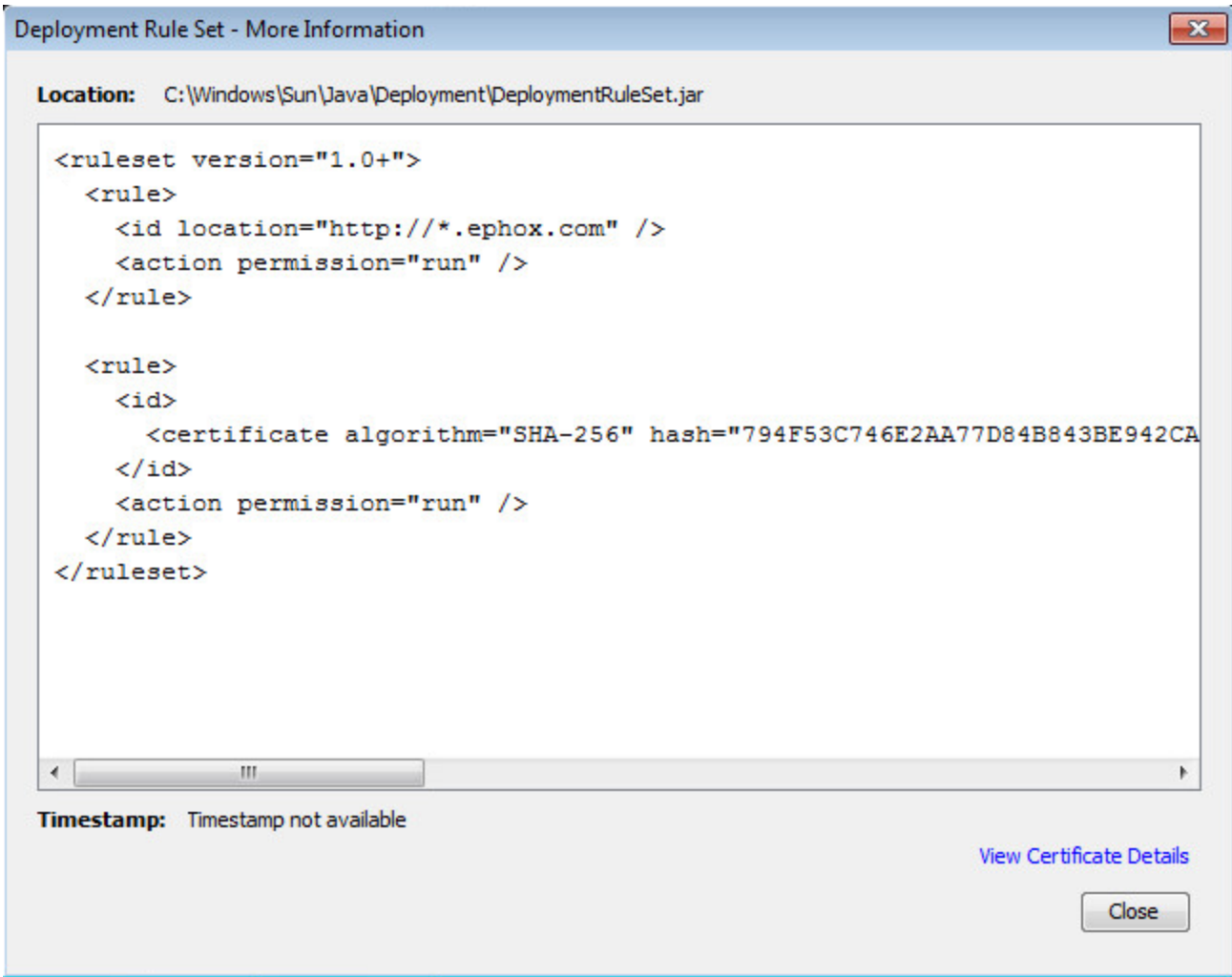
[http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment\\_rules.html#package](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/deployment_rules.html#package)

## Verify the Deployment Rule Set JAR Installation

Once installed to the appropriate location, the **Security** tab in the Java Control Panel should show a "View the active Deployment Rule Set" link, as shown below:

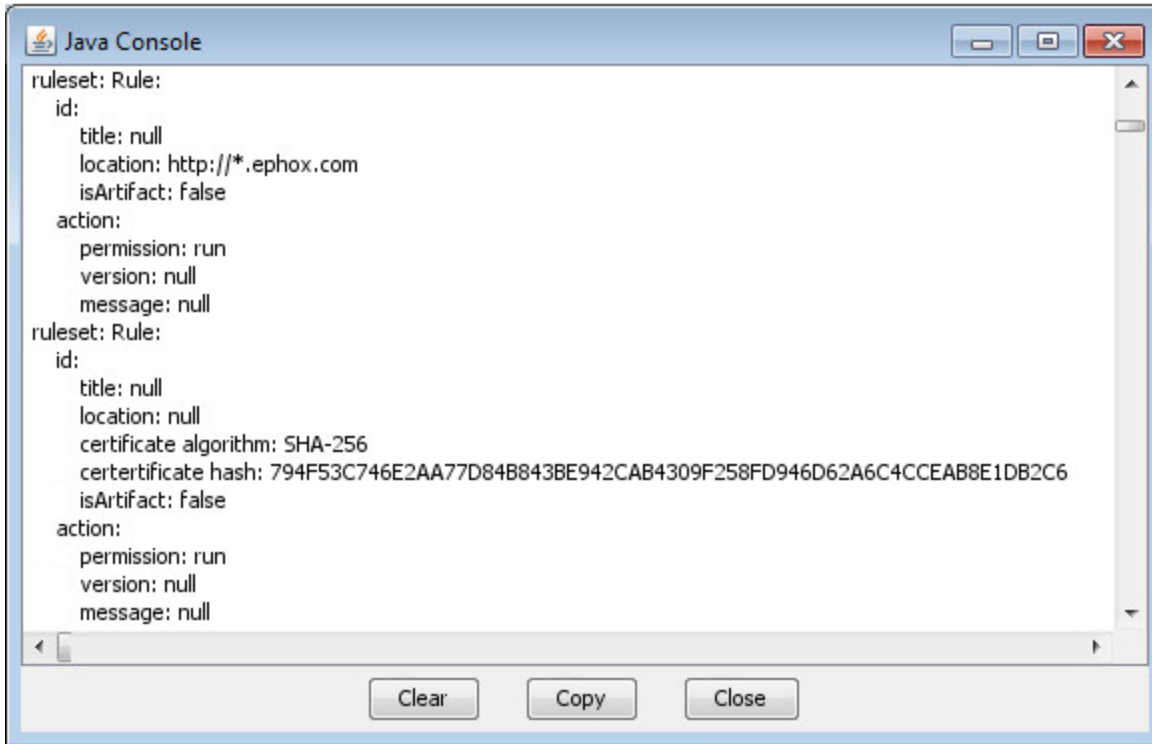


Clicking this link will open an information dialog:

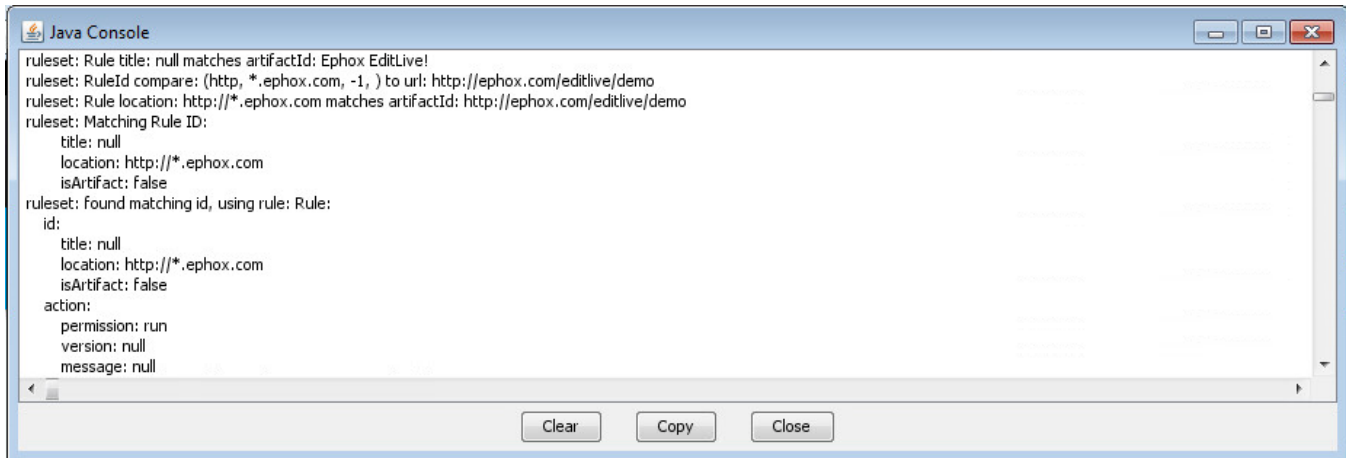


If the **Location** value matches the Rule Set that was installed previously and the displayed text matches the rule, the installation has been completed successfully.

Upon opening the page with the applet, the JVM will read the Rule Set:



The JVM will then match the rule to the running instance of EditLive!:



EditLive! should now run without showing any security dialogs.



# Java SWT Framework

The EditLive Swing SDK can run as a component in a Java Swing UI application. SWT is an alternative UI framework for Java.

EditLive does not directly support SWT, however we know some customers have had success using it.

One customer had success using this wrapper component:

<http://www.eclipse.org/articles/Article-Swing-SWT-Integration/files/example-javadoc/swingintegration/example/EmbeddedSwingComposite.html>

# Developer Guide

- [Instantiating, Configuring, and Using EditLive!](#)
    - [Instantiating, Configuring, and Using the Applet](#)
      - [Instantiating the Applet](#)
      - [Default Values of Load Time Properties](#)
      - [Setting EditLive! Contents](#)
      - [Retrieving Content From EditLive!](#)
      - [Advanced APIs](#)
        - [Introduction to the Advanced APIs](#)
        - [Java APIs](#)
        - [Coding with the Advanced APIs](#)
        - [Configuration File Differences](#)
        - [Compiling and Running the Advanced APIs](#)
    - [Using Inline Editing](#)
      - [Inline Editing Known Issues](#)
    - [Using TinyMCE](#)
      - [TinyMCE Known Issues](#)
  - [Instantiating, Configuring, and Using the Java Swing SDK](#)
    - [Instantiating EditLive! for Java Swing in a Java Application](#)
    - [Java Swing APIs](#)
    - [Compiling Applications](#)
    - [Setting EditLive! For Java Swing Content](#)
    - [Retrieving Content From EditLive! for Java Swing](#)
    - [Java runtime settings](#)
  - [Encoding Content for Use with EditLive!](#)
  - [Manually Editing Configuration Files](#)
  - [Licensing EditLive!](#)
  - [Autosave](#)
  - [Automatic Hyperlinking](#)
  - [Insert HTML Fragment](#)
  - [Java Webstart for Java 9+](#)
- [Enterprise Edition](#)
  - [Enabling Enterprise Edition](#)
  - [Enterprise Edition Features](#)
- [Editor Appearance](#)
  - [Setting Menu and Toolbar Items](#)
    - [Mnemonics and Shortcuts for Menus](#)
  - [Menu and Toolbar Item List](#)
    - [File Commands](#)
    - [Edit Commands](#)
    - [View Commands](#)
    - [Insert Commands](#)
    - [Format Commands](#)
    - [Tool Commands](#)
    - [Table Commands](#)
    - [Properties Commands](#)
    - [Help Commands](#)
    - [Form Commands](#)
    - [Track Changes Commands](#)
    - [Image Editor Commands](#)
    - [Accessibility Commands](#)
    - [Broken Hyperlink Report](#)
    - [Commenting Commands](#)
    - [Equation Editor Commands](#)
    - [Menu and Toolbar Item Groups](#)
  - [Creating Custom Menu and Toolbar Items](#)
  - [Customizing the Color Picker](#)
- [Cascading Stylesheet Support](#)
  - [Using CSS in EditLive!](#)
    - [Using CSS in the Applet](#)
    - [Using CSS in the Swing SDK](#)
  - [Using CSS Extensions to Render Custom Tags](#)
  - [Restricting what CSS classes appear in EditLive! styles drop down](#)
- [Proofing and Language Tools](#)
  - [Tiny Dictionaries](#)
  - [Creating, Modifying and Adding to Dictionaries](#)
  - [Tiny Thesauruses](#)
  - [Auto Correct Spelling](#)
- [Image and Media Support](#)
  - [Working with Images](#)
    - [Image Editing](#)
    - [HTTP Upload Support for Images and Objects](#)
      - [ASP.NET Upload Script](#)
      - [ASP Upload Script](#)
      - [Cold Fusion Upload Script](#)
      - [JSP Upload Script](#)
      - [PHP Upload Script](#)

- Drag and Drop
- Working with Media
  - Using Social Media and External Media Services
  - Using a Media Aggregator Service
  - Embedding Media Using HTML5
- Advanced Media Support
  - Object Tag Support
  - Using WebDAV with EditLive!
  - Enabling WebDAV on a Web Server
  - Image Insertion Dialog's Browser Component
- Collaboration
  - Getting Started With Track Changes
  - Track Changes Serialization Format
  - Commenting
- Web Content Accessibility
  - Accessibility Compliance
  - Accessibility As You Type
  - Table Accessibility
- Internationalization Support
  - Internationalization Support Overview
  - Character Sets Supported
  - Specifying Character Sets for Internationalization
    - Specifying Character Sets in the Applet
    - Specifying Character Sets in the Swing SDK
- Read Only Content and Custom Tags
  - Creating Read Only Content
  - Using Custom Tags
- Equation Editor
  - Integrating the Tiny Equation Editor
- Troubleshooting
  - Load Time Troubleshooting
    - Load Time Troubleshooting in the Applet
    - Load Time Troubleshooting in the Swing SDK
  - Run Time Troubleshooting
  - Minimizing an EditLive! Deployment
  - Optimizing Load Time
  - Managing Crashes
- Plugins
  - Creating and Using Plugins in the Applet
  - Creating and Using Plugins in the Swing SDK
- HTML5 Support
  - Disabling HTML5 Features

# Instantiating, Configuring, and Using EditLive!

- [Instantiating, Configuring, and Using the Applet](#)
  - [Instantiating the Applet](#)
  - [Default Values of Load Time Properties](#)
  - [Setting EditLive! Contents](#)
  - [Retrieving Content From EditLive!](#)
  - [Advanced APIs](#)
    - [Introduction to the Advanced APIs](#)
    - [Java APIs](#)
    - [Coding with the Advanced APIs](#)
    - [Configuration File Differences](#)
    - [Compiling and Running the Advanced APIs](#)
  - [Using Inline Editing](#)
    - [Inline Editing Known Issues](#)
  - [Using TinyMCE](#)
    - [TinyMCE Known Issues](#)
- [Instantiating, Configuring, and Using the Java Swing SDK](#)
  - [Instantiating EditLive! for Java Swing in a Java Application](#)
  - [Java Swing APIs](#)
  - [Compiling Applications](#)
  - [Setting EditLive! For Java Swing Content](#)
  - [Retrieving Content From EditLive! for Java Swing](#)
  - [Java runtime settings](#)
- [Encoding Content for Use with EditLive!](#)
- [Manually Editing Configuration Files](#)
- [Licensing EditLive!](#)
- [Autosave](#)
- [Automatic Hyperlinking](#)
- [Insert HTML Fragment](#)
- [Java Webstart for Java 9+](#)

# Instantiating, Configuring, and Using the Applet

- [Instantiating the Applet](#)
- [Default Values of Load Time Properties](#)
- [Setting EditLive! Contents](#)
- [Retrieving Content From EditLive!](#)
- [Advanced APIs](#)
  - [Introduction to the Advanced APIs](#)
  - [Java APIs](#)
  - [Coding with the Advanced APIs](#)
  - [Configuration File Differences](#)
  - [Compiling and Running the Advanced APIs](#)
- [Using Inline Editing](#)
  - [Inline Editing Known Issues](#)
- [Using TinyMCE](#)
  - [TinyMCE Known Issues](#)

# Instantiating the Applet

The configuration of EditLive! is performed at load-time through the EditLive! [Load Time Methods](#) and a EditLive! [Configuration File](#). The [Load Time Methods](#) allow basic settings to be configured for EditLive!, whereas the [Configuration File](#) includes detailed settings for the interface and the behavior of EditLive!. When conflicts arise between the settings in the Configuration File and the Load Time Properties, the configuration file will always take precedence over any other setting.

The one exception to this rule is when specifying the character set to be used by EditLive!. For more information on this feature, see the [Specifying Character Sets in the Applet](#) article.

## Configuration via the Load Time Properties of EditLive!

The EditLive! [Load Time Methods](#) enable developers to configure the properties of EditLive! in addition to setting the content. The Load Time Properties also provide the means to specify the [Configuration File](#) to be used to configure the majority of the settings for EditLive!. These load-time properties are available in JavaScript, ASP and ASP.NET.

### Important Load Time Properties

The following load-time properties are very important based on the context of the functions the developer wishes EditLive! to perform.

- [setAutoSubmit Method](#)  
The property enables or disables the ability for EditLive!'s content to be automatically submitted during HTTP Posts. For more information on this functionality see the article [Retrieving Content From EditLive!](#).
- [setBaseURL Method](#)  
Sets the URL which EditLive! uses to resolve relative URLs in the document.
- [setConfigurationFile Method](#) or [setConfigurationText Method](#)  
These load-time properties are used to specify the configuration file to be used for the specific instance of EditLive!. The properties load a configuration file as either a URL reference to a configuration file or as a string representation of the configuration file XML.
- [setDebugLevel Method](#)  
Sets the level of detail provided in the Java console debug log.
- [setHeight Method](#)  
Sets the height of the EditLive! applet.
- [setReturnBodyOnly Method](#)  
This property stipulates whether to return either the entire contents of EditLive! or only the contents nested between the <body> HTML tags within EditLive!.
- [setWidth Method](#)  
Sets the width of the EditLive! applet.

## Configuration via an EditLive! Configuration File

An EditLive! configuration file is a single XML file used to specify numerous elements of the EditLive! editor. Configuration files facilitate the customization of the behavior and functionality of EditLive!. Almost the entire interface of EditLive! can be customized via the configuration file. Customization of the interface may also include the development of custom functionality accessed via custom toolbar and menu items.

Because server-side languages can be used to generate documents of any type at run-time, configuration files can also be stored as server-side language files (e.g. JSP files, ASP files) as long as the files contain calls to render the information as XML at run-time.

EditLive! requires a configuration file to load. For information on how to specify a configuration file to load with an instance of EditLive!, please see the [setConfigurationFile Method](#) and [setConfigurationText Method](#) articles.

In cases where either the settings in the document content of EditLive! or the settings in the load-time properties conflict with the settings in the configuration file, the settings in the configuration file will take precedence. For example, if the configuration file specifies style

```
H1{font-size: 10;}
```

and the [setStyles Method](#) for an instance of EditLive! specifies

```
H1{font-size: 20} H2{font-size: 15}
```

, the resulting embedded style for the XHTML will be

```
H1{font-size: 10}
```

Configuration files can be created and edited by manually editing an EditLive! Configuration File via text editor.

## Instantiating EditLive! Example

The following example demonstrates how to use the EditLive! load-time properties to instantiate the editor, as well as specify the EditLive! configuration file to be used.

```
<script src="../../redistributables/editlivejava/editlivejava.js"/>

<form name="form1" method="POST">
  <script type="text/javascript">
    var editlivel;
    editlivel = new EditLiveJava("ELApplet1", "700", "400");
    editlivel.setConfigurationFile("sample_eljconfig.xml");
    // ...
    // other load time properties specified here as well
    // ...
    editlivel.show();
  </script>
</form>
```

## See Also

- [Load Time Methods](#) of EditLive!
- [EditLive! Configuration File Elements](#)

# Default Values of Load Time Properties

## Default Values for EditLive! Load Time Properties

The following provides a list of the default values for the load time properties of EditLive!.

[Load Time Methods](#) are available in JavaScript, ASP and ASP.NET. For more information on these languages and their implementation see the [Load Time Methods](#) section of this SDK.

Property Name	Default Value	Language Support
<a href="#">AutoSubmit</a>	true	All Languages
<a href="#">DebugLevel</a>	info	All Languages
<a href="#">Height</a>	400 (pixels)	ASP, ASP.NET
<a href="#">HttpLayerManager</a>	default	All Languages
<a href="#">LocalDeployment</a>	false	All Languages
<a href="#">ReturnBodyOnly</a>	true	All Languages
<a href="#">ShowSystemRequirementsError</a>	true	All Languages
<a href="#">Width</a>	700	ASP, ASP.NET



# Setting EditLive! Contents

There are several methods in which XHTML content can be loaded into an instance of the EditLive! applet:

- Using the [Open Menu and Toolbar Items](#)
- Using the [setDocument Method](#)
- Using the [setBody Method](#)

There are a variety of methods that can be used to set the CSS style information for an instance of EditLive!. For more information on how to set the CSS information for an instance of EditLive!, please see the [Using CSS in the Applet](#) article.

## Using the Open Menu and Toolbar Items

Using the EditLive! [Configuration File](#), developers can specify the Open dialog to be accessed via either a menu item or toolbar button. This dialog allows users to specify an XHTML document load into an instance of EditLive!.

These menu and/or toolbar items can be added to an EditLive! Configuration File by [manually editing the configuration file](#). If manually editing a configuration file, see the [Menu and Toolbar Item List](#), `<menuItem>`, and `<toolbarButton>` sections of this SDK.

## Using the Document Load Time Property

The [setDocument Method](#) allows the developer to specify an entire XHTML document to load into EditLive!. The XHTML document is passed to the Document load time property as a [URL encoded](#) string.

### Example

The following example depicts loading a simple XHTML document into an instance of EditLive! using the Document load time property. Note the use of the Javascript `encodeURIComponent` function to encode the XHTML content.

To ensure the most reliable URL encoding when encoding content for the [setDocument Method](#), you should use the URL encoding method provided by your server-side language. For a list of available URL encoding methods for a variety of server-side languages, see the [Encoding Content](#) article.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.setDocument(encodeURIComponent("<html><body><p>Test Content</p></body></html>"));
  ...
  editlivejava1.show();
</script>
```

## Using the Body Load Time Property

The [setBody Method](#) allows the developer to specify only the content to be loaded between the XHTML `<body>` tag in EditLive!. The XHTML content is passed to the Body load time property as a [URL encoded](#) string.

### Example

The following example depicts loading simple XHTML content into the `<body>` tag of an instance of EditLive!, using the Body load time property. Note the use of the Javascript `encodeURIComponent` function to encode the XHTML content.

To ensure the most reliable URL encoding when encoding content for the Body load time property, you should use the URL encoding method provided by your server-side language. For a list of available URL encoding methods for a variety of server-side languages, see the [Encoding Content](#) article.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.setBody(encodeURIComponent("<p>Test Content</p>"));
  ...
  editlivejava1.show();
</script>
```

# Retrieving Content From EditLive!

There are several methods in which content can be retrieved from an instance of the EditLive! applet:

- Using the **Save** and **Save As...** Menu and Toolbar Items.
- Using the EditLive! Run Time Functions  
EditLive! provides run-time functions to allow access to an editor's content at any time.
- Through Hidden Form Fields Created During a HTTP Post.  
When the HTML form containing an instance of EditLive! is submitted, several hidden form fields are generated containing content from the editor. These fields are available when:
  - The *onsubmit* function of the form is called and client-side scripting can be used to access the hidden form fields before the form is posted, or
  - The form is submitted and server-side scripting can be used to access the fields sent through HTTP Post.
- Allowing EditLive! to Explicitly Call a HTTP Post.  
EditLive! can explicitly call a HTTP Post (as opposed to the HTML form containing the instance of EditLive! calling a HTTP Post). To do this, you will need to use a custom created menu or toolbar item.

## Using the Save and Save As... Menu and Toolbar Items

Using the **Save** and **Save As...** menu and toolbar items, users can save the entire XHTML document stored in EditLive! to their local machine.

These items are specified through an EditLive! configuration file. These items can be added by [Manually Editing Configuration Files](#). See the [Menu and Toolbar Item List](#), `<menuitem>` and `<toolbarButton>` sections of this SDK for more information.

## Using the EditLive! Run Time Functions

EditLive! provides two run-time functions to extract the content of the editor:

- [getDocument Method](#)  
This property returns the entire XHTML document currently stored in EditLive!.
- [getBody Method](#)  
This property returns only the content nested between the `<body>` tags in EditLive!. The actual `<body>` tags are not included.

Both of these functions take two parameters:

- A string matching a JavaScript function's name. This JavaScript function will be called upon invoking one of the above run-time properties, passing the content of EditLive! as a parameter.
- An (optional) boolean value, indicating whether the [uploadImages Method](#) should be invoked before the above JavaScript function is called. This will ensure all locally stored images are uploaded to the relevant Web server and their URLs are adjusted in the EditLive! content.

### Example

This example shows how to use the [getDocument Method](#).

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.show();
</script>

<!--
  Pressing the button causes the content to be displayed in an alert.
-->
<input type="button" name="GetContent" onclick="alert(editlivejava1.GetDocument(true));"/>
```

## Using the HTML Form HTTP Post

By default, the content of the EditLive! applet are retrieved when a HTML form containing the instance of EditLive! is submitted. Because of the lack of LiveConnect support on various operating systems and browsers, EditLive! populates a hidden field with its contents automatically rather than the developer calling for the contents explicitly. The name of this generated hidden field (contained within the same HTML form as the EditLive! instance) is given the same name that was specified by the developer when the EditLive! instance was created.

### Example

This example shows an instance of EditLive! being created with the name ELApplet1.

```

<form name="form1" method="POST" action="http://www.yourserver.com/scripts/upload.jsp">
  <script src="../../redistributables/editlivejava/editlivejava.js"></script>

  <script language="JavaScript">
    var editlivejava1;
    editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
    ...
    editlivejava1.show();
  </script>
</form>

```

If *ELApplet1* was specified as the name for the instance of the EditLive! applet, then the applet would store its contents in the hidden field named *ELApplet1*. This hidden field is then posted with the rest of the form data when the submit button is pressed.

When a form containing an instance of EditLive! is submitted, a hidden field containing all style information for the document is also created and posted with the form data. This hidden field will be called *ELAPPLETNAME\_styles*, where *ELAPPLETNAME* is the name specified by the developer when the instance of EditLive! was created.

EditLive! automatically updates the hidden fields by attaching itself to the form's **onsubmit()** handler. If there is already a function specified in the **onsubmit()** handler, then this function will run after the hidden field has been updated. This means that you can still use the **onsubmit()** handler to run your own JavaScript functions. If you use another button/image/event to submit the form by calling **form.submit()**, the browser will not call the **onsubmit()** handler and EditLive! will not populate the hidden fields with data. For this reason, please ensure you use **form.onsubmit()** to avoid this problem.

By default, the content that is retrieved from the EditLive! applet will be the contents of the <BODY> element for the HTML Document. This behavior can be modified through setting the [setReturnBodyOnly Method](#).

The automatic submission of the content of the EditLive! applet can be disabled through the use of the [setAutoSubmit Method](#). By setting the AutoSubmit property to *false*, the automatic content submission can be disabled. Automatic submission of content by the applet is enabled by default.

## Retrieving EditLive!'s Content Using Client-Side Scripting Before the HTTP Post

By using the **onsubmit()** handler for the HTML form containing EditLive! client-side scripting can be used to access the hidden field containing EditLive!'s content.

If a HTML form's **onsubmit()** handler returns *false*, the HTML form won't submit. This technique can be used to extract the content of EditLive! without submitting the HTML form.

### Example

The following example shows how to use the **onsubmit()** handler for a HTML form (containing an instance of EditLive!) to extract EditLive!'s content. In this example, the **onsubmit()** handler also cancels the HTML form submission using *return false*.

```

<form name="form1" method="POST" onsubmit="return retrieveELContents()">

  <script src="../../redistributables/editlivejava/editlivejava.js"></script>

  <script language="JavaScript">
    var editlivejava1;
    editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
    // ensuring a field is created in the form called ELApplet1, containing the contents of
    editlivejava1.setAutoSubmit(true);
    ...
    editlivejava1.show();
  </script>

  <script language="JavaScript">
    function retrieveELContents()
    {
      // Setting contents of hidden field's value to a JavaScript variable
      var ELContents = document.getElementById('ELApplet1').value;

      alert(ELContents);

      return false;
    }
  </script>

  <input type="submit" name="submit" value="Submit"/>
</form>

```

The **showBodyOnly** attribute of the EditLive! [<sourceEditor>](#) Configuration File Element defines how the editor will store its contents in the hidden form field(s) identified above. If showBodyOnly is set to *true*, only the contents of the <BODY> element (for the HTML document stored in EditLive!) will be stored in the hidden form field. If showBodyOnly is set to *false*, the entire HTML document stored in EditLive! will be stored in the hidden form field.

## Retrieving EditLive!'s Content Using the HTTP Post's Server-Side Script

### Example

The following examples show using a server-side language to access the hidden HTML form field generated by instance of EditLive! named *ELApplet1*.

#### VB Scripting

```
<%  
    Dim editorContent = Request( "ELApplet1" )  
%>
```

#### JSP Scripting

```
<%  
    String editorContent = request.getParameter("ELApplet1");  
%>
```

## Using EditLive! to Explicitly Call a HTTP Post

EditLive! can be configured to post its content directly to a Post Acceptor Script on a Web server. This is useful in situations where EditLive! cannot post its content as part of the HTML form submission architecture.

This technique creates two separate methods for retrieving the content of the EditLive! for Java editor:

- [Accessing the content via the Post Acceptor Script](#).  
Examples of how to access the content sent from EditLive! to the Post Acceptor Script is given in the Retrieving EditLive! for Java's Content Using the HTTP Post's Server-Side Script section of this article.
- Specifying either a **Save As..** dialog or a JavaScript function to be called upon the HTTP Response. How to specify what action is performed upon the HTTP Response is different for the [Custom Menu Item](#) and [Custom Toolbar Button](#).

EditLive! for Java allows a HTTP Post to be explicitly called through the creation of a [Custom Menu Item](#) or a [Custom Toolbar Button](#) with a PostDocument action.

## Important Considerations When Using An Explicit EditLive! HTTP Post

Developers must consider several things when using the EditLive! applet to make the HTTP POST instead of using the browser's submit mechanisms:

- If it is required that other form variables be submitted with the content of EditLive!, it is best to avoid using the explicit HTTP functionality of EditLive!. This is because when using the EditLive! explicit HTTP Post, the post of EditLive! occurs separately to the browser's post. This can unnecessarily complicate the server-side processing involved in receiving and processing the posted content.
- The Post functionality can be used to submit the content of EditLive! to a completely different Post Acceptor Script than the browser's post. Thus, EditLive! can submit its content easily to two separate processes. This can be useful when using EditLive! in association with a related server-side process which produces a response which should be saved to the client.

## Ensuring Output is XHTML or XML Compliant

In order to ensure that the output of content in EditLive! is XHTML or XML compliant, specific attributes in the [<htmlFilter>](#) configuration file element have to be set.

For XHTML compliant output, the following filter settings are required:

- *Set outputXHTML to true* - This ensures that XHTML tags are used (i.e. <br/> instead of <br>).
- *Set allowUnknownTags to false* - This ensures that no tags outside of the XHTML standard are used, i.e. custom tags. Instead, custom tags are HTML encoded.
- *Set encloseText to true* - This ensures content is correctly nested inside the relevant parent tags.

For XML compliant output the following filter settings are required:

- *Set outputXML to true* - this ensures that special characters are encoded as numeric entities and that XML style tags are used (i.e. <br/> instead of <br>).
- *Set encloseText to true* - This ensures that the content is correctly nested inside the relevant parent tags.

See Also

- [getBody Method](#)
- [getDocument Method](#)
- [uploadImages Method](#)
- [<customMenuItem>](#) Configuration File Element
- [<customToolBarButton>](#) Configuration File Element
- [<htmlFilter>](#) Configuration File Element

# Advanced APIs

- [Introduction to the Advanced APIs](#)
- [Java APIs](#)
- [Coding with the Advanced APIs](#)
- [Configuration File Differences](#)
- [Compiling and Running the Advanced APIs](#)

# Introduction to the Advanced APIs

EditLive!'s [Advanced API](#) and [plugin](#) functionality are only supported with an EditLive! [Enterprise Edition](#) license.

Unlike the standard APIs for EditLive!, which are used in the webpage loading the applet, the advanced APIs are Java classes and interfaces specifying the default functionality of the EditLive! applet. These advanced APIs are used to extend the core functionalities of the applet. The instantiation APIs can then be used when the applet is loaded into a webpage to specify information such as the document and styles loaded into EditLive!.

To use the advanced APIs developers must create a new Java class. In a webpage loading an instance of EditLive!, the following load-time properties can be used to specify this new class to load instance of the default instance of EditLive!:

- [addJar method](#)
- [addPlugin method](#)
- [addPluginAsText method](#)

Any other values specified through the [Load Time Methods](#) (e.g. EditLive! configuration file, the XHTML document to be loaded into the editor) are then sent to this newly defined class upon loading EditLive!.

The following steps outline how the advanced APIs are used to customize EditLive!:

1. A Java class needs to be created that accepts an instance of the [ELJBean](#) class as its only parameter in its constructor. Within this class the developer then uses the methods of [ELJBean](#) to define the appearance and functions of the EditLive! applet.
2. This class is then compiled and packaged in a jar file.
3. To load this newly defined instance of EditLive! instead of the default applet instance, use the [addJar](#), [addPlugin](#), or [addPluginAsText](#) load-time methods. All other values defined using the EditLive! [Load Time Methods](#) will also be passed to this new Java class.

## Use Cases

The Advanced APIs are used when developers want to customize EditLive! beyond the supported functionality of the [Load Time Methods](#), [Run Time Methods](#), and [Configuration File Elements](#). Some example use cases are:

- Creating Java swing dialogs to appear based on custom buttons/toolbars being pressed.
- Overriding the current hyperlink and image dialogs to display a custom created dialog.
- Creating a customized rendering for specific elements and custom tags.

# Java APIs

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

The Advanced APIs are a set of Java classes and interfaces, allowing extensive customization of the EditLive! applet. The classes used to customize the applet are documented in the [EditLive! JavaBean APIs](#). These APIs are documented in Java-docs style, detailing the properties and methods of each interface and class used in the advanced APIs.

The [Java APIs](#) can be accessed in Java-doc format in the EditLive! SDK.

For the [ELJBean](#) class in the Advanced APIs, the [ELJBean](#) constructor cannot be used. To use the [Advanced APIs](#), a Java class must be defined where the constructor takes an instance of [ELJBean](#) as it's parameter.

In order to load the newly created Java class utilizing the Advanced APIs, you can use either the [AddJar](#), [AddPlugin](#), or [AddPluginAsText](#) load-time property.



# Coding with the Advanced APIs

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

This article provides developers with an understanding of how the Advanced APIs for EditLive! operate. This article also details the basic steps required by developers to extend the core functionality of an EditLive! applet by creating Java classes that utilize the Advanced APIs.

## How the Advanced APIs Work

To load an instance of EditLive! in a webpage, developers use the [Load Time Methods](#) and a [Configuration File](#). The Advanced APIs allow developers to then create a Java class that is passed to an instance of the EditLive! applet with all the properties defined in the above mentioned [Load Time Methods](#) and the [Configuration File](#). This developer-designed Java class can then be used to access the properties of the EditLive! applet and specify additional functionality for the applet.

## Coding with the Advanced APIs

In order to use the [Advanced APIs](#) to extend the functionality of EditLive!, developers must create their own Java class. This Java class needs to utilize the Tiny Java APIs in order to access the properties of the EditLive! applet.

Java classes created to use the [Advanced APIs](#) need to adhere to the following properties:

- Import the required Java packages to utilize the [Advanced APIs](#).

The following packages are required to use the Tiny [Java APIs](#):

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
```

- Ensure the Java class' constructor has only one parameter: An instance of ELJBean.

The ELJBean class is the Java representation of the EditLive! applet. All the information specified in applet's Load Time Properties and Configuration File are passed to the ELJBean class. This allows developers to not only specify additional functionality for the EditLive! applet, but also access the current properties of the EditLive! applet as well. For information on how to interact with the ELJBean class see the Tiny Java APIs.

### Example

The following example shows how to specify a Java class called *AdvancedAPITest* that can utilize the [Advanced APIs](#).

```
public class AdvancedAPITest {
    ELJBean _bean;

    public AdvancedAPITest(ELJBean bean) {
        _bean = bean;

        // perform actions on _bean using Tiny Java APIs
    }
}
```

For information on how to compile and run Java classes utilizing the [Advanced APIs](#), see the [Compiling and Running the Advanced APIs](#) article.

## See Also

- [Compiling and Running the Advanced APIs](#)
- [Tiny Java APIs](#)

# Configuration File Differences

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

Instances of EditLive! using either the Advanced APIs or the default editor both need to reference an EditLive! Configuration File. For a complete list of all the configuration file elements available to EditLive!, please see the [Configuration File Elements](#) article.

Using the Advanced APIs, developers can customize in Java the actions to occur on clicking custom toolbar and menu items. The following documentation extends on the EditLive! configuration file elements depicted in the [Configuration File Elements](#) article:

- [<customMenuItem>](#)
- [<customToolBarButton>](#)
- [<customComboBoxItem>](#)

## <customMenuItem> Configuration File Element

This element specifies the properties for a developer-defined custom menu item for use within EditLive!.

### Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<customMenuItem.../>
```

```
<editLive>
...
<menuBar>
...
<menu>
  <customMenuItem... />
</menu>
...
</menuBar>
...
</editLive>
```

### Required Attributes

#### name

The name which uniquely defines the custom menu item.

#### text

The text to place on the menu for this item.

#### action

The action which this menu item performs when clicked on. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this configuration file. The following values will be assigned to the Java TextEvent sent with the event:
  - *Action Command* - *TextEvent.CUSTOM\_ACTION*
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom menu item was selected.

The event sent to the bean will also have the value *TextEvent.CustomAction.RAISE\_EVENT* added to the extra *int* property of the event.

When the event has been handled (by using the event method *setHandled(boolean b)*, where *b = true*), the javascript function defined in value will be called.

- *customPropertiesDialog* - Fires an event for the [ELJBean](#) using this configuration file. The following values will be assigned to the TextEvent sent with the event:
  - *Action Command* - *TextEvent.SHOW\_CUSTOM\_PROPERTIES*
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom menu item was selected.

#### value

The value of this attribute depends on the value specified in the action attribute:

- *insertHTMLAtCursor* - value will be a string of HTML

- *InsertHyperlinkAtCursor* - value will be a URL
- *raiseEvent* - value will be the name of a JavaScript function
- *customPropertiesDialog* - value will be the Extra String field of the TextEvent sent with the [ELJBean](#) event.

When using the `insertHTMLAtCursor` **action** the HTML to be inserted must be URL encoded in the configuration file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3C/p%3E`.

## Optional Attributes

### imageURL

The URL of the image to be placed on the menu with the menu item text. The image should be of a .gif format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide.

This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! is embedded.

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

## Examples

The following example demonstrates how to define a custom menu item for use within EditLive!. The menu item defined in this example will insert HTML at the cursor; note that the value in the example below is URL encoded.

```
<editLive>
...
<menuBar>
...
<menu name="Example">
  <customMenuItem
    name="customItem1"
    text="Custom Item"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="insertHTMLAtCursor"
    value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
  </menu>
...
</menuBar>
...
</editLive>
```

The following example demonstrates how to define a custom properties dialog which is launched from a custom menu item for use within EditLive!. The custom properties dialog will be available for use when the cursor is inside any `<td>` tag.

```
<editLive>
...
<menuBar>
...
<menu name="Example">
  <customMenuItem
    name="customProperties1"
    text="Custom td Properties"
    action="customPropertiesDialog"
    value="customTDFunction"
    enableintag="td" />
  </menu>
...
</menuBar>
...
</editLive>
```

## Remarks

The `<customMenuItem>` element can appear multiple times within the `<menu>` element.

The `<customMenuItem>` element must be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<customMenuItem name=... />
```

Text assigned to the value attribute must be [URL encoded](#) as it is in the example above.

## <customToolBarButton> Configuration File Element

This element will cause a particular button to be present on the toolbar within Tiny EditLive!

### Configuration Element Tree Structure

```
<editLive>  
<toolbars>  
<toolbar>  
<customToolBarButton.../>
```

```
<editLive>  
  ...  
  <toolbars>  
    ...  
    <toolbar>  
      <customToolBarButton ... />  
    </toolbar>  
  ...  
</toolbars>  
  ...  
</editLive>
```

#### Required Attributes

##### **name**

The name which uniquely defines the custom toolbar button.

##### **text**

The tooltip text for this custom toolbar button.

##### **action**

The action which this toolbar button performs when clicked on.

Note: This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this configuration file. The following values will be assigned to the Java TextEvent sent with the event:
  - *Action Command* - `TextEvent.CUSTOM_ACTION`
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom toolbar button was clicked.

The event sent to the bean will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra *int* property of the event.

When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the javascript function defined in value will be called.

- *customPropertiesDialog* - Fires an event for the [ELJBean](#) using this configuration file. The following values will be assigned to the TextEvent sent with the event:
  - *Action Command* - `TextEvent.SHOW_CUSTOM_PROPERTIES`
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom toolbar button was clicked.

##### **value**

The value of this attribute depends on the value specified in the **action** attribute:

- *insertHTMLAtCursor* - value will be a string of HTML
- *InsertHyperlinkAtCursor* - value will be a URL
- *raiseEvent* - value will be the name of a JavaScript function
- *customPropertiesDialog* - value will be the Extra String field of the TextEvent sent with the ELJBean event.

When using the `insertHTMLAtCursor` action the HTML to be inserted must be URL encoded in the configuration file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3C/p%3E`.

**imageUrl**

The URL of the image to be placed on the toolbar button. The image should be of a .gif format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide.

This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! is embedded.

## Optional Attributes

**enableintag**

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

## Examples

The following example demonstrates how to define a custom toolbar button for use within EditLive!. The button defined in this example will insert HTML to insert at the cursor; note that the value in the example below is URL encoded.

```
<editLive>
...
<toolbars>
...
  <toolbar name="Example">
    <customToolbarButton
      name="customItem1"
      text="Custom Item"
      imageUrl="http://www.someserver.com/image16x16.gif"
      action="insertHTMLAtCursor"
      value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </toolbar>
  ...
</toolbars>
...
</editLive>
```

The following example demonstrates how to define a custom properties dialog which is launched from a custom toolbar button for use within EditLive!. The custom properties dialog will be available for use when the cursor is inside any `<td>` tag.

```
<editLive>
...
<toolbars>
...
  <toolbar name="Example">
    <customToolbarButton
      name="customProperties1"
      text="Custom td Properties"
      imageUrl="http://www.someserver.com/image20x20.gif"
      action="customPropertiesDialog"
      value="customTDFunction"
      enableintag="td" />
    </toolbar>
  ...
</toolbars>
...
</editLive>
```

## Remarks

The `<customToolbarButton>` element can appear multiple times within the `<toolbar>` element.

The `<customToolbarButton>` element must be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<customToolbarButton name=... />
```

Text assigned to the value attribute must be [URL encoded](#) as it is in the example above.

**<customComboBoxItem> Configuration File Element**

This element specifies the properties for a developer defined custom combo box item for use within Tiny EditLive!. The custom combo box item must be listed within a `<customToolbarComboBox>` element and will therefore appear on one of the toolbars within EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolbarComboBox>
<customComboBoxItem .../>
```

```
<editLive>
...
<toolbars>
...
<toolbar>
  <customToolbarComboBox>
    <customComboBoxItem .../>
  </customToolbarComboBox>
</toolbar>
...
</toolbars>
...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom combo box item within the `<customToolbarComboBox>` element. This means that there cannot be two `<customToolbarComboBoxItem>` elements with the same name within one `<customToolbarComboBox>` element.

### text

The text to represent this item within the combo box it is to be listed in.

### action

The action which this menu item performs when clicked on.

Note: This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this configuration file. The following values will be assigned to the Java `TextEvent` sent with the event:
  - *Action Command* - `TextEvent.CUSTOM_ACTION`
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom combo box item was clicked.

The event sent to the bean will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event.

When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the javascript function defined in value will be called.

- *customPropertiesDialog* - Fires an event for the [ELJBean](#) using this configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - *Action Command* - `TextEvent.SHOW_CUSTOM_PROPERTIES`
  - *Extra String* - The string specified in the **value** attribute
  - *Extra Object* - A Java Map containing the attributes of the element selected when the custom combo box item was clicked.

### value

The value of this attribute depends on the value specified in the **action** attribute:

- *insertHTMLAtCursor* - value will be a string of HTML
- *insertHyperlinkAtCursor* - value will be a URL
- *raiseEvent* - value will be the name of a JavaScript function
- *customPropertiesDialog* - value will be the Extra String field of the `TextEvent` sent with the [ELJBean](#) event.

Note: When using the `insertHTMLAtCursor` **action** the HTML to be inserted must be URL encoded in the configuration file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3C/p%3E`.

## Examples

The following example demonstrates how to define a custom combo box item for use within a custom combo box which exists on the EditLive! Command Toolbar. The combo box item defined in this example will insert HTML to insert at the cursor; note that the value in the example below is URL encoded.

```
<editLive>
...
<toolbars>
...
<toolbar name="command">
  <customToolbarComboBox name="customCombo">
    <customComboBoxItem
      name="customItem1"
      text="Custom Item"
      action="insertHTMLAtCursor"
      value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </customToolbarComboBox>
  </toolbar>
...
</toolbars>
...
</editLive>
```

### Remarks

The `<customComboBoxItem>` element can appear multiple times within the `<customToolbarComboBox>` element.

The `<customComboBoxItem>` element must be a complete tag, it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<customComboBoxItem name=... />
```

Text assigned to the value attribute must be [URL encoded](#) as it is in the example above.

# Compiling and Running the Advanced APIs

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

The Advanced APIs provide the ability for developers to extend the functionality of EditLive! by accessing the program's underlying Java code.

In order to use the Advanced APIs, developers must first create a Java class utilizing the Tiny EditLive! [Java APIs](#). For information on how to do this see the [Coding with the Advanced APIs](#) article.

After creating the Java class, developers need to perform the following steps:

- Compiling the Java class and storing the required .class files in a Java .jar file.
- Instantiating an instance of EditLive!, using the [AddJar](#), [AddPlugin](#), or [AddPluginAsText](#) load-time property to specify the new Java class to load.

## Compiling Advanced API Java Files

Before compiling a Java file utilizing the Advanced APIs, you must ensure that the *editlivejava.jar* file is on the classpath so that the EditLive! classes can be found. The *editlivejava.jar* file is located in the *INSTALL\_HOME/redistributables/editlivejava* directory, where *INSTALL\_HOME* is the location where the EditLive! SDK has been installed to. The method for doing this will vary depending on your operating system and development environment. If you are using the command-line tools provided with the Java SDK, you can add the option `-classpath /path/to/editlivejava.jar` where */path/to/editlivejava.jar* is the path to the *editlivejava.jar* file.

For example, to compile the *BasicELJ.java* file if the files *BasicELJ.java* and *editlivejava.jar* are both inside the */bin* directory of your Java SDK:

```
javac -classpath ./editlivejava.jar BasicELJ.java
```

This will produce the file *BasicELJ.class*.

All the required .class files compiled for an example then need to be packaged in a .jar file. For example, to package the *BasicELJ.class* file into a jar called *BasicELJ.jar*, if the files *BasicELJ.class* and *editlivejava.jar* are both inside the */bin* directory of your Java SDK:

```
jar cvf BasicELJ.jar BasicELJ.class
```

Created jar files now need to be signed. To sign a jar file you will need to use the Java [jarsigner](#) command. For detailed information on using the jarsigner command of Java see the [Sun Microsystems website](#).

For example, for a private key file *yourkey.p12*, of a keystore type *pkcs12*, with a store password of *yourpassword*, a key password of *yourpassword*, and the alias for the private key *keyuser*, the following command would sign the *BasicELJ.jar* file.

```
jarsigner -keystore "yourket.p12" -storetype "pkcs12" -storepass "yourpassword" -keypass "yourpassword" BasicELJ.jar "keyuser"
```

Please consult the documentation included with your development environment or the Java SDK for more information.

## Running Advanced API Java Files

After compiling a Java file that utilizes the Ephox EditLive! Advanced APIs, the [AddJar](#), [AddPlugin](#), or [AddPluginAsText](#) load time property of EditLive! can be used to specify this new Java class to load when instantiating the EditLive! applet.

### Example

The following example shows how to load the Java class *NewApplet.class*, stored in the jar *MyApplet.jar*, when instantiating an instance of the EditLive! applet. For this example, *MyApplet.jar* would be located in the same directory as the webpage loading EditLive!.

This example specifies the Advanced API implementation of EditLive! by using the [AddJar](#) load-time property.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.addJar("MyApplet.jar", "NewApplet");
  editlivejava1.show();
</script>
```



### Example

The following example shows how to load the Java class *NewApplet.class*, stored in the jar *MyApplet.jar*, when instantiating an instance of the EditLive! applet. For this example, *MyApplet.jar* and *MyPlugin.xml* would be located in the same directory as the webpage loading EditLive!.

This example specifies the Advanced API implementation of EditLive! by using the [AddPlugin](#) load-time property.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.addPlugin("MyPlugin.xml");
  editlivejava1.show();
</script>
```

*MyPlugin.xml* would define the location of the *MyApplet.jar* file and its main class as follows:

```
<?xml version="1.0" ?>
<plugin>
  <advancedapis jar="MyApplet.jar" class="NewApplet" />
</plugin>
```

### Example

The following example shows how to load the Java class *NewApplet.class*, stored in the jar *MyApplet.jar*, when instantiating an instance of the EditLive! applet. For this example, *MyApplet.jar* and *MyPlugin.xml* would be located in the same directory as the webpage loading EditLive!.

This example specifies the Advanced API implementation of EditLive! by using the [AddPluginAsText](#) load-time property.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

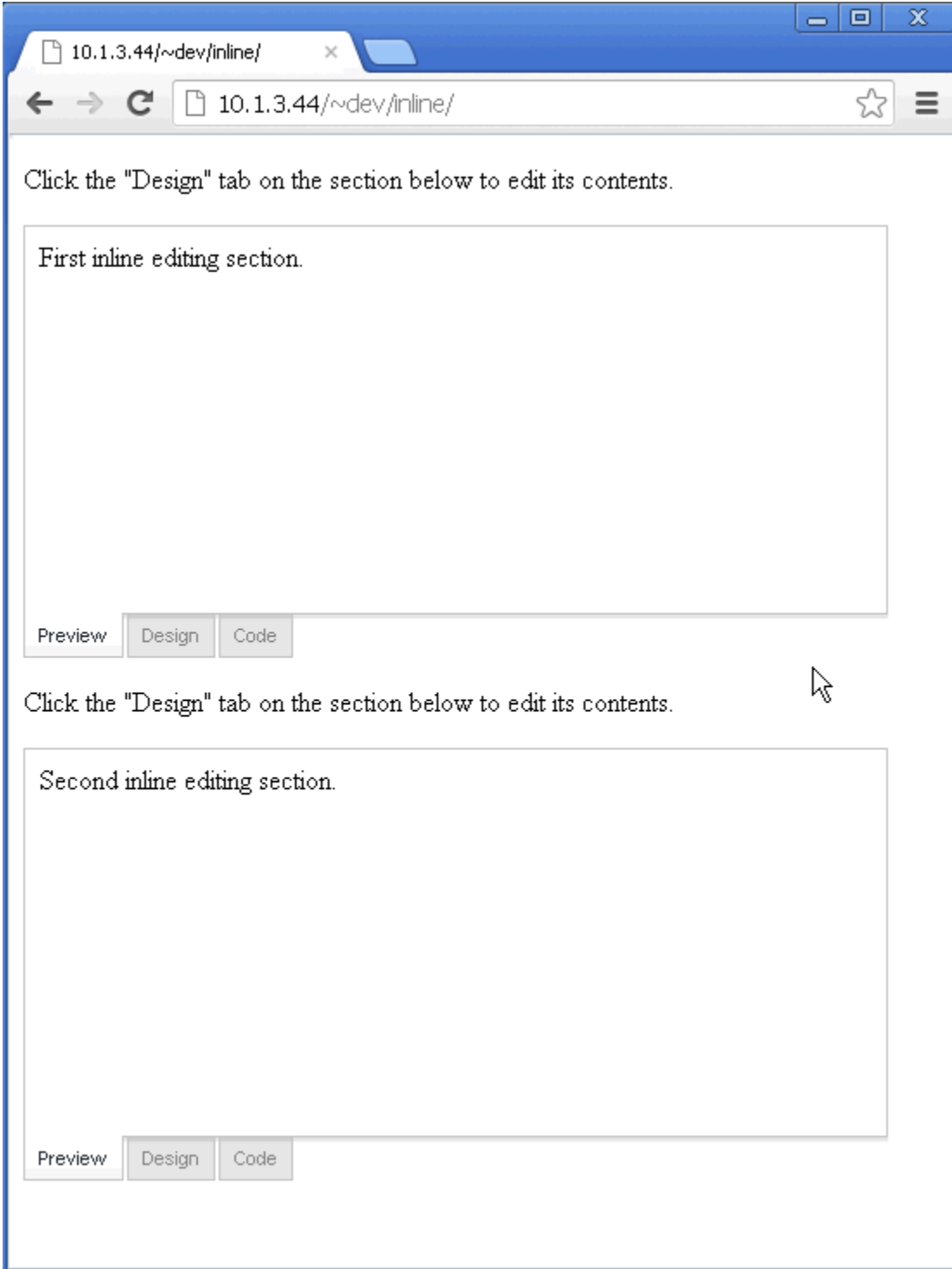
<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
  ...
  editlivejava1.addPluginAsText("<?xml version=\"1.0\" ?><plugin><advancedapis jar=\"MyApplet.jar\" class=\"
NewApplet\" /></plugin>");
  editlivejava1.show();
</script>
```

## See Also

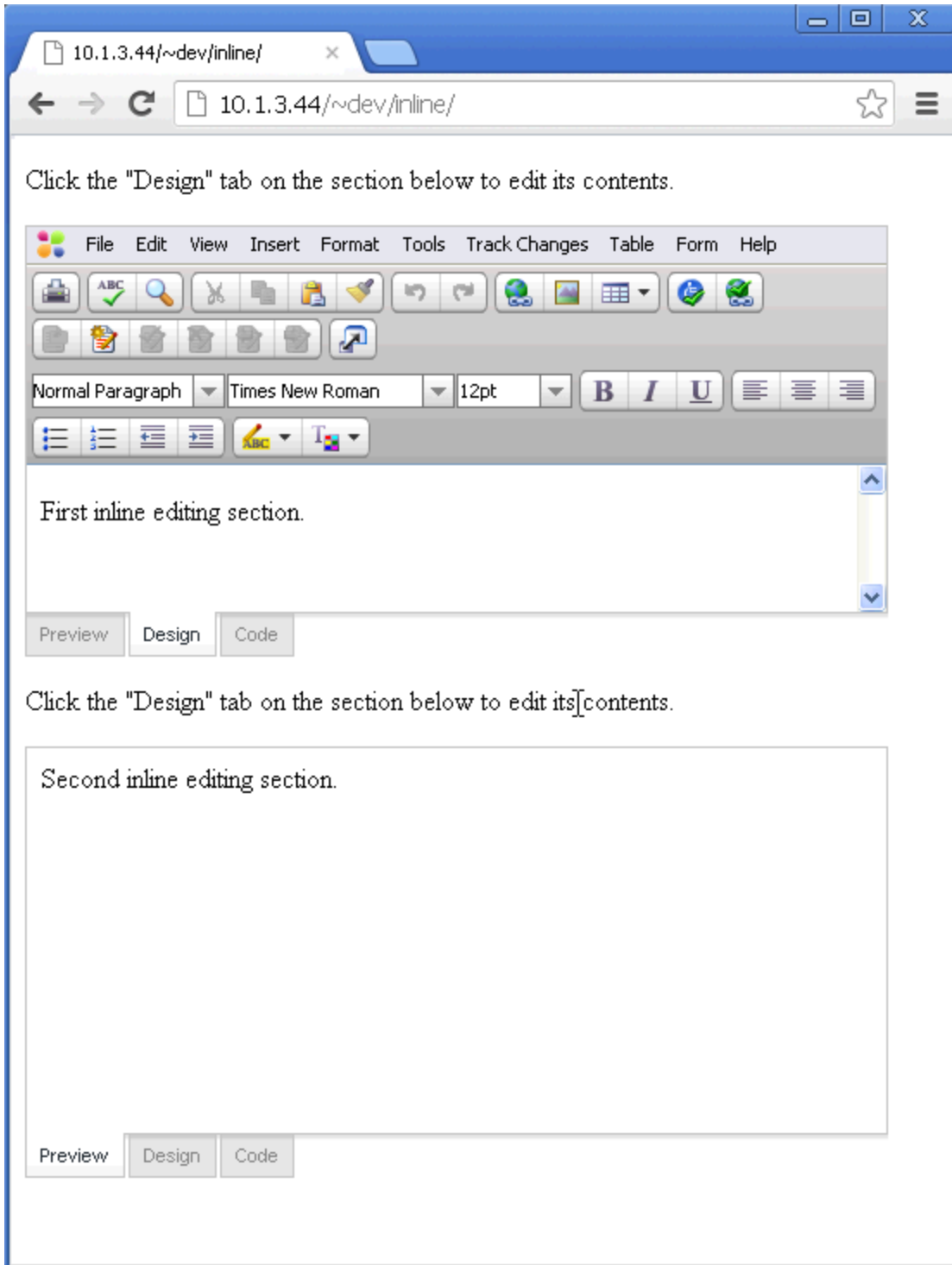
- [Coding with the Advanced APIs](#)
- [addJar Method](#)
- [addPlugin Method](#)
- [addPluginAsText Method](#)

# Using Inline Editing

Inline Editing refers to the functionality available in EditLive! for using a single instance of the editor to generate HTML for several rich text fields. Developers can create any number of DIV HTML elements within their web page and register these DIVs with EditLive!. If a user clicks on a DIV registered with EditLive!, the editor will appear in place of the DIV, loading the DIV's content ready for editing.



Two DIVs registered as Inline Editing Sections with EditLive!.



The result of clicking the first DIV's 'Display' tab.

You can find detailed instructions on how to integrate the Inline Editing implementation of EditLive! in the [Using Inline Editing Tutorial](#) packaged with this SDK.

## Reasons for Using Inline Editing

- Less system resources will be consumed by only loading one instance of EditLive! for Java on the page.
- EditLive! will only be loaded when required by the user (i.e. when a registered DIV is clicked).
- Provides an easy method for integrating EditLive! into several rich text areas occurring in a webpage.
- Easier facilitation for inline editing.

## Creating Inline Editing Sections

Note: The ASP.NET integration of EditLive! requires a different implementation of Inline Editing. Please consult the following articles for more information:

- [InlineEditing Property \(ASP.NET only\)](#) for ASP.NET
- [Inline Editing ASP.NET Example](#)

To integrate the Inline Editing implementation of EditLive! into your webpage, you'll need to perform the following:

1. Define the required EditLive! load-time properties in your webpage (i.e. [setConfigurationFile/setConfigurationText](#), [setDownloadDirectory](#)).
2. Specify the *inlineEditing.js* file in your code (this JavaScript file is located in the same directory as the *editlivejava.js* file packaged with the EditLive! SDK).
3. Create DIV HTML elements in your page you wish to become inline editing sections. Ensure each DIV has a unique ID attribute.
4. After creating the DIV HTML element, use the [addEditableSection method](#) for EditLive! to register the DIV as an Inline Editing Section.

### Example

```
...  
  
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>  
  
...  
  
<div id="div1" style="height: 550px;" ><p>div content</p></div>  
  
...  
  
<script language="Javascript">  
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");  
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");  
    editlivejs.addEditableSection("div1");  
</script>  
  
...
```

The following example depicts how the [addEditableSection Method](#) should ideally be defined:

```
<html>  
    <head>  
        <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>  
        <script language="Javascript">  
            var editlivejs;  
            function loadEditLive() {  
                editlivejs = new EditLiveJava("ELApplet", "100%", "100%");  
                editlivejs.setConfigurationFile("../../redistributables/editlivejava  
/sample_eljconfig.xml");  
                editlivejs.addEditableSection("div1");  
            }  
        </script>  
    </head>  
  
    <body onload="loadEditLive()">  
        <div id="div1" style="height: 550px;"><p>div content</p></div>  
    </body>  
</html>
```

## Populating Inline Editing Sections with Content

An Inline Editing section can be populated with either a HTML fragment or an entire HTML document. There are two methods for populating an Inline Editing section:

- Insert content into the TEXTAREA
- [setContentForEditableSection method](#)

### Insert Content into the TEXTAREA

To use this implementation you will need to create a TEXTAREA HTML element with an ID attribute containing a corresponding Inline Editing DIV ID with *\_contentArea* appended.

### Example

This example creates a hidden TEXTAREA HTML element which will allow users to edit the HTML document contained in the TEXTAREA via Inline Editing in the DIV with the ID *div1*. The Inline Editing section will display the content `<html><body><p>HTML document for editing.</p></body></html>`.

Note: any HTML content in the TEXTAREA needs to be HTML Encoded - i.e. any characters with special meaning in HTML must be escaped to their entity encoding, e.g. `<` becomes `&lt;`;

```
<html>
  <head>
    <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
    <script language="Javascript">
      var editlivejs;
      function loadEditLive() {
        editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
        editlivejs.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejs.addEditableSection("div1");
      }
    </script>
  </head>

  <body onload="loadEditLive()">
    <div id="div1" style="height: 550px;"><p>This content will not be seen, the content inside the
body tags of the text area below will be displayed to the user instead.</p></div>
    <textarea id="div1_contentArea" style="display: none;">
      &lt;html&gt;
        &lt;body&gt;
          &lt;p&gt;HTML document for editing.&lt;/p&gt;
        &lt;/body&gt;
      &lt;/html&gt;
    </textarea>
  </body>
</html>
```

### SetContentForEditableSection Run-Time Property

To manually insert HTML content into an Inline Editing Section (whether it is represented by an instance of EditLive! or the DIV itself), use the [setContentForEditableSection Method](#). A HTML fragment or an entire HTML document can be passed with this method.

## Retrieving Content from Inline Editing Sections

The contents of an Inline Editing Section can be retrieved in one of two ways:

- Using the GetContentForEditableSection Run-Time Function
- Using a HTML Form HTTP Post

### Using the GetContentForEditableSection Run-Time Function

The [getContentForEditableSection Method](#) allows you to extract the contents of a specific Inline Editing Section.

You can combine the [getEditableSections](#) and [getContentForEditableSection](#) run-time functions to iterate through each Inline Editing Section and extract their respective contents.

### Example

```
<script language="Javascript">
  var editableSectionArray = editlive_js.getEditableSections();

  for(i = 0; i < editableSectionArray.length; i++) {
    alert("Editable Section DIV ID: " + editableSectionArray[i] + ", Contents: " + editlive_js.
getContentForEditableSection(editableSectionArray[i]));
  }
</script>
```

### Using a HTML Form HTTP Post

If each registered DIV is nested within a HTML FORM element, the contents of each Inline Editing Section will be passed to the FORM's post handler script. The contents of each Inline Editing Section will be passed to the post handler script via a hidden TEXTAREA element with a NAME attribute matching the ID attribute for the Inline Editing Section DIV.

### Example

A DIV is nested in a HTML FORM element. This DIV is registered as an Inline Editing Section with EditLive!. The DIV has an ID value of *div1*. When the FORM is submitted, the contents of the DIV will be passed to the post handler script via HTTP as the POST argument *div1*.

```
...
<body>
  <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
  <form name="form1" action="posthandler.jsp" method="post">
    ...
    <div id="div1" style="height: 550px;"></div>
    ...
  </form>
  <script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("div1");
  </script>
</body>
...
```

### See Also

- [addEditableSection Method](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [Using Inline Editing Tutorial](#)

# Inline Editing Known Issues

This article outlines several known issues with the Inline Editing functionality of EditLive! and solutions for overcoming these issues.

## Inline Editing Sections Disappearing

Clicking on a DIV registered as an Inline Editing Section may cause the DIV to disappear from the page completely. The most likely reason for this issue occurring is that the DIV does not have a fixed height specified. Any DIV registered as an Inline Editing Section with EditLive! requires a fixed height.

### Example

The following example depicts an a DIV element that does not have a fixed height specified.

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="div1">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("div1");
</script>
```

If a user clicked in *div1*, it would disappear from the page.

### Example

The following code creates a DIV with a height of 550 pixels. This DIV is then registered as an Inline Editing Section.

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="div1" style="height: 550px;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("div1");
</script>
```

## Clicking on an Inline Editing Section Does Not Load EditLive!

This issue can occur for several reasons. One of the most common ones can be solved thusly:

- Ensure the parameter passed to the [addEditableSection Method](#) is a case-sensitive match for your specified DIV's ID attribute.

## Local Images Not Appearing in Inline Editing Sections


For some users, local images inserted into EditLive! may not render when switching between Inline Editing Section DIVs.

### Example


The following screenshots depict a local image inserted into EditLive!. Once the user clicks on another Inline Editing Section (hence removing EditLive! from it's current Inline Editing Section), the local image does not display in the DIV.

File Edit View Insert Format Tools Table Form Track Changes

Normal Paragraph Arial 10pt **B** *I* U

Local image:  **Ephox**<sup>®</sup> (ephoxlogo.gif, 137x43 pixels).

Design Code

The image shows a screenshot of a web editor interface. At the top, there is a menu bar with options: File, Edit, View, Insert, Format, Tools, Table, Form, and Track Changes. Below the menu is a toolbar with various icons for editing and formatting. The main editing area shows a paragraph of text: "Local image:  **Ephox**<sup>®</sup> (ephoxlogo.gif, 137x43 pixels)." The text is in a bold font. Below the text, there is a mouse cursor icon. At the bottom of the editor, there are two tabs: "Design" and "Code".



Local image: (*ephoxlogo.gif*, 137x43 pixels).



This issue occurs due to security implementations in several browsers that restrict local images being displayed in web pages. This issue can be overcome by ensuring you have configured EditLive! to upload any local images to a server-side location. You can find information on how to configure EditLive! to upload local images in the [HTTP Upload Support](#) article in this SDK.

If an upload script has been specified for EditLive!, this script will be invoked every time the user switches between Inline Editing Sections.

## Issues with EditLive! Run Time Functions

When utilizing EditLive!'s [Run Time Methods](#) in conjunction with the inline editing implementation of the editor, ensure no run time functions for the editor are invoked unless the editor has completed loading into an Inline Editing Section. If a run time function for the editor is invoked and EditLive! is still loading, JavaScript errors can occur and the run time function will not be performed.

## Browser Freezes When Switching Between Inline Editing Sections

### Mac Browsers (Safari and Firefox)

Users may find that switching between multiple inline sections over an extended period may cause the browser to crash. Java tends to run out of memory due to poor memory management in Java on the Mac. Using Inline Editing for Mac is preferable to using many instances of the editor in a single page. Using several instances of the editor in a single page will only make the memory usage issue worse.

### Firefox on Windows Vista and Linux Operating Systems

Users may experience browser freezes if they quickly switch between inline editing sections for an extended time (constantly switching between sections around 30-40 times). Normal usage on Vista and Linux allows Java to reclaim memory properly and avoid browser freezing.

## Issues with Mac Browsers (Safari and Firefox)

### Difficulties Obtaining Editor Focus Using Safari

Occasionally, the shifting of focus between the editor and the browser gets confused. On these occasions, switching focus from the editor to another DIV requires two clicks (one to give the browser back into focus and another to click on the DIV) instead of one. This can also cause the highlighting of inline editing sections to stop working until the browser re-attains focus.

## Inline Editing Sections Rendering with an Unexpected Height

If inline editing section DIVs feature differing heights, EditLive! for Java will always render using the largest height specified by an inline editing section DIV.

### Example

In the following example, a HTML page contains two inline editing sections. The first section has a specified height of 400 pixels and the second has a height of 600 pixels. If the user were to click in the first inline editing section, the editor would render with a height of 600 pixels despite the DIV's fixed height of 400 pixels.

```
...
<body>
  <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
  <form name="form1" action="posthandler.jsp" method="post">

    ...

    <div id="div1" style="height: 400px;"></div>

    ...

    <div id="div2" style="height: 600px;"></div>

  </form>
  <script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("div1");
    editlivejs.addEditableSection("div2");
  </script>
</body>
...
```

## EditLive! Takes Several Seconds to Load Into an Inline Editing Section

When using Firefox 2.0 in combination with JRE 1.6, loading the editor into an inline editing section may take several seconds.

## <FORM> Elements in EditLive! Content Cause Unpredictable Page Behaviour

Currently the Inline Editing functionality for EditLive! does not support content loading that contains <form> elements. If content is specified for EditLive! Inline Editing sections that contains <form> elements, the following problems can occur depending on your browser:

- Clicking an Inline Editing section may not load the EditLive! for Java editor.
- Upon loading EditLive!, the <form> element and all it's contents may be stripped out of EditLive!'s HTML.

## Inline Editing Does Not Support TinyMCE

Inline Editing currently does not support the [TinyMCE Editor](#). If Inline Editing is enabled, the [setExpressEdit](#) property will be ignored and EditLive! will continue to be used in the inline editing sections.

## See Also

- [addEditableSection Method](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [Using Inline Editing Tutorial](#)

# Using TinyMCE

**Select Edit works with TinyMCE 3.x only.**

The TinyMCE JavaScript editor is supported by Tiny. While EditLive! contains more extensive and powerful functionality than TinyMCE, TinyMCE does not require Java to be installed on the user's machine.

In order to allow developers to easily integrate either EditLive! or TinyMCE into their application, the EditLive! APIs have been extended to easily facilitate loading either editor. These extensions to the EditLive! APIs are referred to as the Select Edit APIs.

Before attempting to integrate TinyMCE into your application using the API calls below, please consult the [Installing TinyMCE](#) article in the Install Guide component of the EditLive! for Java SDK.

If you are upgrading from EditLive! 7.1, you will need to upgrade the legacy Tiny Express Edit JavaScript editor with TinyMCE. You will need to follow the [Installing TinyMCE](#) instructions and the integration steps listed below.

## Integrating TinyMCE using the Select Edit APIs

To enable TinyMCE in your application, take a standard implementation of EditLive! and perform the following steps:

1. Reference the `expressEdit/tinymce/jscripts/tiny_mce/tiny_mce.js` file located in your `redistributables/editlivejava` directory.

```
...
<script language="Javascript" src="../../redistributables/editlivejava/expressEdit/tinymce/jscripts
/tiny_mce/tiny_mce.js"></script>
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
...

<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", 640, 400);
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.show();
</script>
...
```

2. Add a call to the [SetExpressEdit](#) load-time API. In the example below, TinyMCE will load as long as the client's platform and browser is compatible with TinyMCE. (For more information on TinyMCE compatibility, please see the [TinyMCE Compatibility Chart](#))

```
...
<script language="Javascript" src="../../redistributables/editlivejava/expressEdit/tinymce/jscripts
/tiny_mce/tiny_mce.js"></script>
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
...

<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", 640, 400);
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.setExpressEdit("always");
    editlivejs.show();
</script>
...
```

## Setting Styles and Configuring Toolbar Buttons for TinyMCE

When using the Select Edit APIs to load TinyMCE, the styles and toolbar buttons specified in the EditLive! configuration file will also be applied to the TinyMCE instance. Not all toolbar buttons available to EditLive! will be applicable for TinyMCE. Please consult the [TinyMCE Known Issues](#) article for a complete list of TinyMCE-compatible EditLive! toolbar buttons.

If no style information has been specified in the EditLive! configuration file, the Select Edit APIs will search for style information from the **head** element of the document loaded into TinyMCE. If no style information exists here, style information will be loaded via the [SetStyles property](#).

## Setting and Retrieving Content from TinyMCE

The majority of load-time and run-time APIs in EditLive! for Java are supported when integrating TinyMCE (e.g. GetBody, SetBody). For a comprehensive list of supported APIs please consult the [TinyMCE Known Issues](#) article.

## See Also

- [setExpressEdit Method](#)
- [TinyMCE Known Issues](#)

# TinyMCE Known Issues

**Select Edit works with TinyMCE 3.x only.**

This article outlines several known issues with Tiny support for TinyMCE.

## Supported Load-Time Properties

TinyMCE supports the following load-time properties in the [Select Edit APIs](#):

- The EditLive [JavaScript Constructor](#)
- [setDownloadDirectory Method](#)
- [setConfigurationFile Method](#)
- [setDocument Method](#)
- [setBody Method](#)
- [show Method](#)
- [setExpressEdit Method](#)

## Supported Run-Time Functions

TinyMCE supports the following run-time functions in the [Select Edit APIs](#):

- [getBody Method](#)
- [getDocument Method](#)
- [getSelectedText Method](#)
- [insertHTMLAtCursor Method](#)
- [insertHyperlinkAtCursor Method](#)
- [setBody Method](#)
- [setDocument Method](#)

## TinyMCE Does Not Support AutoSubmit

[Select Edit APIs](#) do not currently support the [setAutoSubmit Method](#).

## TinyMCE Does Not Support Image Uploading









Image uploading is not supported when using the [Select Edit APIs](#) to instantiate TinyMCE in much the same way as EditLive!. Calling [getBody Method](#) or [getDocument Method](#) will not upload images. The [uploadImages Method](#) is not available for the Select Edit API instantiation of TinyMCE.


## Supported Toolbar Buttons

Although the same EditLive! configuration file can be used to load either EditLive! or TinyMCE, not all of the toolbar buttons available to EditLive! will appear in TinyMCE.



The following list of EditLive! toolbar buttons are supported by TinyMCE, although being a different editor TinyMCE's equivalent toolbar buttons will present the functionality in a different manner. Not all of the shortcut keys for toolbar buttons in EditLive! will carry over to TinyMCE.



For more information on specifying toolbar buttons in your EditLive! configuration file, please consult the [Setting Menu and Toolbar Items](#) article.






Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image
Create a new file.	New	New	CTRL+N	
Save a file to the local machine.	Save	Save	CTRL+S	
Print the contents of EditLive!	Print	Print...	N/A	
Display EditLive! in a separate window.	Popout	Window View	N/A	
Cut the selection.	Cut	Cut	CTRL+X	
Copy the selection.	Copy	Copy	CTRL+C	
Paste.	Paste	Paste	CTRL+V	
List of styles available for use in this document.	Style	N/A	N/A	N/A
List of fonts available for use in this document.	Face	N/A	N/A	N/A
List of font sizes available for use in this document.	Size	N/A	N/A	N/A
Bold text.	Bold	Bold	CTRL+B	

Italic text.	Italic	Italic	CTRL+I	
Underline text.	Underline	Underline	CTRL+U	
Strikethrough text.	Strike	Strikethrough	N/A	
Remove formatting.	RemoveFormatting	Remove Formatting	CTRL + SPACE	
Insert a horizontal line.	HRule	Horizontal Line	N/A	
Insert a symbol.	Symbol	Symbol...	N/A	
Find text in the editor.	Find	Find...	CTRL+F	
Undo the last editor action.	Undo	Undo	CTRL+Z	
Redo the last undone editor action.	Redo	Redo	CTRL+Y	
Insert a bookmark.	Bookmark	Bookmark...	N/A	
Insert a hyperlink.	HLink	Insert Hyperlink...	CTRL+K	
Remove a hyperlink.	RemoveHyperlink	Remove Hyperlink	N/A	
Insert Image	ImageServer	Insert Image...	N/A	
Insert an embedded object or multimedia file.	InsertObject	Insert Object...	N/A	N/A
Increase the paragraph or list indent.	IncreaseIndent	Increase Indent	N/A	
Decrease the paragraph or list indent.	DecreaseIndent	Decrease Indent	N/A	
Load EditLive! Help	showHelp	Help...	N/A	
Show or hide paragraph markers and editing grid lines.	ParagraphMarker	Show/Hide Paragraph Markers	N/A	
Delete the current table.	DelTable	Delete Table	N/A	
Edit the selected row's properties.	PropRow	Row Properties...	N/A	N/A
Delete a column from a table.	DelCol	Delete Column	N/A	
Edit the current cell's properties.	PropCell	Cell Properties...	N/A	N/A
Merge cells in a table.	Merge	Merge Cells	N/A	
Split a cell in a table.	Split	Split Cell...	N/A	
Delete a row from a table.	DelRow	Delete Row	N/A	

Any of the three following toolbar buttons will insert an Insert Table toolbar button into TinyMCE. TinyMCE will only display a single Insert Table toolbar, regardless of how many instances of the buttons below are specified in the configuration file.

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image
Insert a table wizard. Allows user to click and select number of rows and columns.	InsTableWizard	Insert Table	N/A	
Insert a table.	InsTable	Insert Table...	N/A	
Edit the current table's properties.	PropTable	Table Properties...	N/A	N/A

Function	XML Name Attribute (Group Item)	XML Name Attribute (Individual Item)	Menu or Tool Tip Text	Shortcut	Image
Insert an ordered list or change an unordered list to an ordered list.	List	OrderedList	Ordered List	N/A	
Insert an unordered list or change an ordered list to an unordered list.	List	UnorderedList	Unordered List	N/A	
Set left alignment.	Align	AlignLeft	Align Left	CTRL+L	

					
Set center alignment.	Align	AlignCenter	Align Center	CTRL+E	
Set right alignment.	Align	AlignRight	Align Right	CTRL+R	
Superscript text.	Script	Superscript	Superscript	N/A	
Subscript text.	Script	Subscript	Subscript	N/A	

The following EditLive! toolbar buttons create several TinyMCE toolbar buttons.

Function	EditLive! Toolbar XML Name Attribute	Generated TinyMCE Toolbar Buttons
Insert a Date and Time.	datetime	Insert Date
Insert a Date and Time.	datetime	Insert Time
Paste Special.	pastesimal	Paste as Plain Text
Paste Special.	pastesimal	Paste from Word
Insert a row in the current table.	insrow	Insert Row Before
Insert a row in the current table.	insrow	Insert Row After
Insert a column in the current table.	inscol	Insert Column Before
Insert a column in the current table.	inscol	Insert Column After
Insert rows or columns in the table.	insrowcol	Insert Row Before
Insert rows or columns in the table.	insrowcol	Insert Row After
Insert rows or columns in the table.	insrowcol	Insert Column Before
Insert rows or columns in the table.	insrowcol	Insert Column After

## TinyMCE Will Not Load if Track Changes or Tiny Comments are Specified in the Default Content

If the [setDocument](#) or [setBody](#) properties contain [Track Changes](#) information or [Tiny Comments](#) and [setExpressEdit](#) is set to attempt to display TinyMCE, the editor will not load.

## Track Changes or Tiny Comments Inserted into TinyMCE at Run-Time Will Corrupt Content

If [Track Changes](#) content or [Tiny Comments](#) are inserted into TinyMCE once the editor has loaded (e.g. using the [setBody](#) or [setDocument](#) run-time functions) the content will be corrupted. We strongly advise you do not insert these formats into the editor.

## XML Content and MathML

TinyMCE does not support XML, including most custom tags and MathML. MathML issues can be avoided by using the [createEquationImage](#) attribute of the [<mathml>](#) configuration file element. This generates an [<img>](#) tag instead of a [<mathml>](#) element and will not be corrupted by TinyMCE.

## Configuration Files Can Not be Loaded from the Local File System

When using the [setConfigurationFile](#) property, you need to ensure the specified configuration file is loaded from a webserver. For example, if the page featuring TinyMCE is loaded using a [file:///](#) URL and the argument passed to [setConfigurationFile](#) is relative, the configuration file will not load. Similarly, if the argument passed to [setConfigurationFile](#) uses the [file:///](#) protocol, the configuration file will not load.

## TinyMCE Is Not Supported In Inline Editing

[Inline Editing](#) currently does not support the TinyMCE Editor. If Inline Editing is enabled, the [setExpressEdit](#) property will be ignored and EditLive! will continue to be used in the inline editing sections.

## Closing Inline Dialogs

The "Escape" key can be used to close all inline dialogs (even if there is no visible close button).

# Instantiating, Configuring, and Using the Java Swing SDK

- [Instantiating EditLive! for Java Swing in a Java Application](#)
- [Java Swing APIs](#)
- [Compiling Applications](#)
- [Setting EditLive! For Java Swing Content](#)
- [Retrieving Content From EditLive! for Java Swing](#)
- [Java runtime settings](#)



# Instantiating EditLive! for Java Swing in a Java Application

In order to create an instance of the EditLive! for Java Swing within a Java Application, developers need to utilize several different methods of the Bean to ensure the desired functionality. For detailed information on creating and modifying an instance of the EditLive! for Java Swing see the [Java APIs](#).

## EditLive! for Java Swing Constructors

EditLive! for Java Swing provides several different constructors. The main difference between these constructors is the way in which they allow the EditLive! configuration file to be specified. EditLive! for Java Swing configuration files can also be specified using several different methods of the *ELJBean*. More information on these methods is depicted in the Specifying the Configuration File section of this page.

## Specifying the Configuration File

An EditLive! for Java Swing configuration file is a single XML file used to specify numerous elements of the EditLive! for Java Swing editor. A configuration file facilitates the customization of the behavior and functionality of EditLive!for Java Swing . Almost the entire interface of EditLive! for Java Swing can be customized via the configuration file. Customization of the EditLive! for Java Swing interface may also include the development of custom functionality accessed via custom toolbar and menu items.

Because server-side languages can be used to generate documents of any types at run-time, configuration files can also be stored as server-side language files (e.g. JSP files, ASP files) as long as the file contains calls to render the information as XML at run-time.

In cases where either the settings in the document content of EditLive! for Java Swing or the settings specified through the Swing methods conflict with the settings in the configuration file, the settings in the configuration file will take precedence. For example, if the configuration file specifies style

```
H1{font-size: 10;}
```

and then the [setStyles](#) ELJBean method specifies

```
H1{font-size: 20} H2{font-size: 15}
```

, then the resulting embedded styles for the XHTML will still only be

```
H1{font-size: 10}
```

Configuration files can be created and edited by [manually editing an EditLive! Configuration File](#) using a text editor.

The *ELJBean* class provides the following constructors to allow developers to specify the configuration file:

- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, Document dXML, boolean init\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, File fXML\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, File fXML, boolean init\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, String xXML\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, String sXML, boolean init\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, URL xmlURL\)](#)
- [ELJBean\(String sHTML, String sStyles, int iWidth, int iHeight, URL xmlURL, boolean init\)](#)

The *ELJBean* class also supports specifying a configuration file using the following methods:

- [setConfigurationDOM](#)
- [setConfigurationFile](#)
- [setConfigurationText](#)
- [setConfigurationURL](#)

## Instantiating EditLive! for Java Swing Example

The following example demonstrates how to use the default EditLive! for Java Swing constructor to instantiate the editor, as well as specify the EditLive! configuration file to be used.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class BasicExample {
```

```

public static void main(String[] args) {
    // Create a new EditLive! instance.
    ELJBean elj = new ELJBean(false);

    // Specify Configuration File
    // NOTE: Ensure you have an EditLive! configuration file in the same directory as the BasicExample.
java class.
    elj.setConfigurationURL(BasicExample.class.getResource("configFile.xml").toString());

    // Finally, initialize the bean.
    elj.init();

    // Set up a JFrame and add the bean to it.
    JFrame frame = new JFrame("Basic Example");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    frame.getContentPane().add(elj);

    frame.pack();
    frame.show();
}
}

```

For more information, consult the [Java APIs](#).

See Also

- EditLive! [Configuration File Elements](#)

# Java Swing APIs

EditLive! for Java Swing comes packaged with the a diverse collection of Java classes. These Java classes allow developers to control the functionality of EditLive! as well as capturing and firing a variety of events that operate in conjunction with the editor.

The [Java Swing APIs](#) are located on the [Tiny website](#). These APIs are presented in the standard Java-doc format.

# Compiling Applications

This article gives developers information on how to compile and run Java applications containing an instance of the EditLive! for Java Swing.

## Compiling Applications Containing the EditLive! for Java Swing

Before compiling a Java application containing the EditLive! for Java Swing, you must ensure that the *editlivejava.jar* file is on the classpath so that the EditLive! for Java Swing classes can be found. The method for doing this will vary depending on your operating system and development environment. If you are using the command-line tools provided with the Java SDK, you can add the option *-classpath /path/to/editlivejava.jar* where */path/to/editlivejava.jar* is the path to the *editlivejava.jar* file.

### Example

To compile the *BasicEL.java* file, if the files *BasicEL.java* and *editlivejava.jar* are both inside the */bin* directory of your Java SDK:

```
javac -classpath .;editlivejava.jar; BasicEL.java
```

This will produce the file *BasicEL.class*.

```
java -classpath .;editlivejava.jar; BasicEL
```

This will run the BasicEL application.

# Setting EditLive! For Java Swing Content

There are several methods in which XHTML content can be loaded into an instance of the EditLive! for Java Swing applet:

- Using the Open Menu and Toolbar Items
- Using the `setDocument()` method for ELJBean
- Using the `setBody()` method for ELJBean

There is also a variety of methods that can be used to set the CSS style information for the instance of EditLive! for Java Swing. For information on how to set the CSS information for an instance of EditLive! for Java Swing, see the [Using CSS in the Swing SDK](#) article.

## Using the Open Menu and Toolbar Items

Using the EditLive! for Java Swing [Configuration File](#), developers can specify the Open dialog to be accessed via either a menu item or toolbar button. This dialog allows users to specify an XHTML document load into the instance of EditLive! for Java Swing.

These menu and/or toolbar items can be added to an EditLive! for Java Configuration File by [manually editing the configuration file](#). If manually editing a configuration file, see the [Menu and Toolbar Item List](#), `<menuItem>` and `<toolbarButton>` sections of this SDK.

## Using the `setDocument()` Method for ELJBean

The `setDocument()` method allows the developer to specify an entire XHTML document to load into EditLive! for Java Swing. The XHTML document is passed to the `setDocument()` method as a Java String.

### Example

The following example depicts loading a simple XHTML document into an instance of EditLive! for Java using the `setDocument()` method.

```
ELJBean editLive = new ELJBean();

String htmlContent = new String("<html><head><title>Example Page</title></head><body><p>This is an example html page</p></body></html>");

editLive.setDocument(htmlContent);
```

## Using the `setBody()` Method for ELJBean

The `setBody()` method allows the developer to specify only the content to be loaded between the XHTML `<body>` tag in EditLive! for Java Swing. The XHTML content is passed to the `setBody()` method as a Java String.

### Example

The following example depicts loading simple XHTML content into the `<body>` tag of an instance of EditLive! for Java Swing, using the `setBody()` method.

```
ELJBean editLive = new ELJBean();

String htmlContent = new String("<p>This is an example html page</p>");

editLive.setBody(htmlContent);
```

# Retrieving Content From EditLive! for Java Swing

There are several methods in which content can be retrieved from an instance of the EditLive! for Java Swing applet:

- Using the **Save** and **Save As...** Menu and Toolbar Items.
- Using the ELJBean methods  
The EditLive! for Java Swing provides methods to allow access to an editor's content at any time.
- Allowing EditLive! for Java Swing to Explicitly Call a HTTP Post.  
EditLive! for Java can explicitly call a HTTP Post using the PostDocument action for a [customToolbarItem](#).

## Using the Save and Save As... Menu and Toolbar Items

Using the **Save** and **Save As...** menu and toolbar items, users can save the entire XHTML document stored in EditLive! for Java Swing to their local machine.

These items are specified through an EditLive! for Java Swing configuration file. These items can be added by [manually editing the configuration file](#). If manually editing a configuration file, see the [Menu and Toolbar Item List](#), `<menuitem>` and `<toolbarButton>` sections of this SDK.

## Using the ELJBean methods

EditLive! for Java provides two methods to extract the content of the editor:

- `getDocument()`  
This property returns the entire XHTML document currently stored in EditLive! for Java Swing. The returned value is a Java String.
- `getBody()`  
This property returns only the content nested between the `<body>` tags in EditLive! for Java Swing. The actual `<body>` tags are not included. The returned value is a Java String.

### Example

This example shows how to use the [getDocument\(\)](#) method.

```
ELJBean editLive = new ELJBean();
...
String editorContents = editLive.getDocument();
System.out.println("Editor contents: " + editorContents);
```

## Using EditLive! for Java Swing to Explicitly Call a HTTP Post

EditLive! for Java Swing can be configured to post its content directly to a Post Acceptor Script on a Web server. This is done by either using the [raiseEvent\(\)](#) method of the ELJBean class, or using the value *PostDocument* for the **action** attribute of a custom menu or toolbar item. This is useful in situations where EditLive! for Java Swing cannot post its content as part of the HTML form submission architecture.

### Using the ELJBean raiseEvent() Method to Generate a HTTP Post

Using the Java APIs packaged with EditLive! for Java Swing, developers can create TextEvents specifying a HTTP Post and the information to send with this post. This TextEvent can then be used with the ELJBean [raiseEvent\(\)](#) method to fire the HTTP Post.

### Example

```
ELJBean editLive = new ELJBean();
...

editLive.raiseEvent(new TextEvent(this, TextEvent.CUSTOM_ACTION, "POST_field##ephox##http://someserver/postacceptor.jsp##ephox##callback##ephox##JSFunction", TextEvent.CustomAction.POST_DOCUMENT));
```

## Retrieving the Content of EditLive! for Java Swing from a HTTP Post

Using HTTP Posts creates two separate methods for retrieving the content of the EditLive! for Java Swing editor:

- Accessing the content via the Post Acceptor Script.  
**Example**  
The following examples show using a server-side language to access the document of an instance of ELJBean. This example assumes the content of ELJBean was sent in a field called *ELContent*. For more information on sending HTTP Posts see the `<customMenuitem>` configuration file element.  
*VB Scripting*

```
<%  
    Dim editorContent = Request( "ELContent" )  
%>
```

### JSP Scripting

```
<%  
    String editorContent = request.getParameter( "ELContent" );  
%>
```

- Specifying either a Save As.. dialog or a JavaScript function to be called upon the HTTP Response. How to specify what action is performed upon the HTTP Response is done through the third parameter sent to the TextEvent constructor. See the [Java APIs](#) for more information.

## Ensuring Output is XHTML or XML Compliant

In order to ensure that the output of content in EditLive! for Java Swing is XHTML or XML compliant, specific attributes in the `<htmlFilter>` configuration file element have to be set.

For XHTML compliant output, the following filter settings are required:

- *Set outputXHTML to true* - This ensures that XHTML tags are used (i.e. `<br/>` instead of `<br>`).
- *Set allowUnknownTags to false* - This ensures that no tags outside of the XHTML standard are used, i.e. custom tags. Instead, custom tags are HTML encoded.
- *Set encloseText to true* - This ensures content is correctly nested inside the relevant parent tags.

For XML compliant output, the following filter settings are required:

- *Set outputXML to true* - this ensures that special characters are encoded as numeric entities and that XML style tags are used (i.e. `<br/>` instead of `<br>`).
- *Set encloseText to true* - This ensures that the content is correctly nested inside the relevant parent tags.

See Also

- [<customMenuItem>](#) Configuration File Element
- [<customToolBarButton>](#) Configuration File Element
- [<htmlFilter>](#) Configuration File Element

# Java runtime settings

In order for EditLive's custom fonts to render correctly, the java command must be invoked with the following parameters:

```
-Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```



# Encoding Content for Use with EditLive!

When working with content that is to be placed in EditLive!, it must be ensured that the relevant content is URL encoded before it is used. This encoding is required so that the content can be used with the JavaScript used to instantiate EditLive!.

When instantiating EditLive! with the ASP or ASP.NET load-time properties, the content placed in the [Style](#), [Content](#), [Document](#) and [Body](#) properties is automatically URL encoded using the appropriate method.

It is recommended that this URL encoding operation be performed on the server side through the use of the appropriate server-side scripting function. It is possible to use the JavaScript **encodeURIComponent** function; this is, however, not recommended as these functions are not certain to provide correct URL encoding across all browsers. JavaScript's **encodeURIComponent** function also encodes content using the UTF-8 character set, which may conflict with the [Character Sets Supported](#) for use with EditLive!.

## Client-side URL Encoding Functions

The following functions provide URL encoding for strings on the client-side. Where possible, it is recommended that a server-side URL encoding function be used as the exact behavior of these functions is browser dependant.

### JavaScript

JavaScript provides the **encodeURIComponent** function. The **encodeURIComponent** function will encode content as UTF-8 characters.

This may conflict with the Character Sets specified for use with EditLive!.

## Server-side URL Encoding Functions

Most scripting languages provide a function to URL encode strings. The following section defines URL encoding functions which can be used with several common server side scripting languages.

### ASP

The URL encoding function which can be using in ASP is **Server.URLEncode**. This can be used in the following manner:

```
Server.URLEncode("this string will be url encoded")
```

### ASP.NET (C#)

The URL encoding function which can be used in ASP.NET is **HttpUtility.UrlEncode**. This class is part of the **System.Web** package. The function can be used in the following manner, with the correct "using" statement:

```
using System.Web; ...
HttpUtility.UrlEncode("this string will be url encoded");
```

### JSP (Java)

The URL encoding method which can be used in JSP and Java classes is the **URLEncoder.encode()** method. The **URLEncoder** class can be found in the **java.net** package. The function can be used in the following manner and the relevant import statements must be included:

```
import java.net.URLEncoder; ...
URLEncoder.encode("this string will be url encoded");
```

### PHP

When using the PHP URL encoding functions, it is important to use the **rawurlencode** function as opposed to the **urlencode** function. The function can be used in the following manner:

```
rawurlencode('this string will be url encoded');
```

It is important to note that the **urlencode** function provides different functionality to the **rawurlencode** function. The **urlencode** function encodes spaces as + symbols which may cause errors with EditLive!.

### ColdFusion

When implementing an EditLive! integration with ColdFusion the URL encoding function which should be used is **URLEncodedFormat**. This function can be used in the following way:

```
urlencodedformat("this string will be url encoded");
```

## Perl

Perl does not include a URL Encode function as part of the standard libraries; developers must write their own. This can be achieved through the use of regular expressions. The following code gives an example on how this may be achieved:

```
#!/usr/bin/perl $encodeString = "this string will be url encoded";  
$encodeString = ~s/([Encoding Content for Use with EditLive!^A-Za-z0-9_\-])/uc;  
sprintf("%%%02x",ord($1))/eg;
```

## Encoding with International Characters

When encoding content for use with EditLive! that contains international characters it may be necessary to specify a character set for use with the server-side encoding method. Please consult your programming language reference for more information on how to specify a character set. It is recommended that UTF-8 character encoding is used in these circumstances, though other character encodings may also be used.

Symptoms of using an incorrect character encoding method with EditLive! are:

- The `&` appears on entering text. This symbol means the current font set being used does not support the character entered.
- The symbol `?` appears on entering text. This means character encoding has been corrupted due to an incorrect character encoding method.

### Example

A document's character set has been defined as UTF-8. The body of the document has been set using the *URLEncoder.encode()* Java function in a JSP page, with the ISO02022JP charset. Because symbols created by the ISO02022JP encoding method are not supported by UTF-8, encoding errors will occur.

The methods available to change the character encoding method used by EditLive! are depicted in the [Specifying Character Set](#) article.

## See Also

- [Specifying Character Set](#)
- [Character Sets Supported](#)

# Manually Editing Configuration Files

Developers can manually create and edit EditLive! configuration files using text editors such as Windows Notepad.

Only developers with XML experience and requirements for server side processing should manually edit EditLive! configuration files.

## Creating and Editing Configuration Files

EditLive! configuration files are XML files. Meta information for the configuration file should be stored in the first line of the file.

If you load your configuration file by URL using `setConfigurationFile` to a URL that is on a different domain to your editing page, the cross-domain AJAX request will fail and EditLive! will be unable to load. We recommend moving the file to the same domain, or switching to `setConfigurationText`.

### Example

To specify the file as XML using character encoding UTF-8:

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

The rest of the configuration file will use XML elements defined by Ephox. These elements are outlined in the EditLive! [Configuration File Elements](#) section of the SDK.

## Configuration File Example

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file customizes and configures EditLive!
TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings
-->
<editlive>
  <!-- Default content for the editor -->
  <document>
    <html>
      <!--
      Default document header
      -->
      <head>
        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->
        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->
        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->
        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
```

```

        text-align: center;
    }
    span.comment {
        border: solid 1px #FFFF00;
        background-color: #FFFFCC;
    }
</style>
-->
</head>
<!--
Default document body. Add content here if you want this to be the default when the editor
loads, although this is better done at runtime.
-->
<body>
</body>
</html>
</document>
<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
    <license
        domain="LOCALHOST"
        key="6FFF-4DC5-EDF4-2486"
        licensee="For Evaluation Only"
        release="8.0"
        type="Evaluation License"
        productivityPack="true"
    />
</ephoxLicenses>
<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
The spellCheck element also includes options to turn autocorrect and spell check as-you-type on
-->
<spellCheck startBackgroundChecking="true" startAutoCorrect="true"/>
<!--
<thesaurus jar="thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->
<!--
Specify HTML filter settings
-->
<htmlFilter
    outputXHTML="true"
    outputXML="false"
    xhtmlStrict="false"
    indentContent="false"
    logicalEmphasis="true"
    quoteMarks="false"
    uppercaseTags="false"
    uppercaseAttributes="false"
    wrapLength="0">
</htmlFilter>
<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
    tabPlacement="bottom"
    brOnEnter="false"
    showDocumentNavigator="false"
    disableInlineImageResizing="false"
    disableInlineTableResizing="false"
    enableTrackChanges="false"
>
<!-- Define Custom Tags actions -->
<!--
<customTags>
    <doubleClickActions>
        <action.../>
    </doubleClickActions>

```

```

</customTags>
-->
<!-- Define additional symbols for the symbol dialog here -->
<!--
<symbols></symbols>
-->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>
<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="clean"/>
<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Excel
-->
<excelImport styleOption="merge_inline_styles"/>
<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>
<!--
Specify plugins to load
-->
<plugins>
  <plugin name="autosave" />
  <plugin name="autolink" />
  <plugin name="insertHTML" />
  <plugin name="rtfpaste"/>
  <plugin name="setBackgroundMode"/>
  <plugin name="spelling" />
  <!-- contains some Enterprise Edition features -->
  <plugin name="tableToolbar" />
  <plugin name="accessibility" />
  <!-- Enterprise Edition only -->
  <plugin name="imageEditor" />
  <plugin name="BrokenHyperlinkReport" />
  <plugin name="commenting" />
  <plugin name="templateBrowser" />
</plugins>
<!--
Specify templates for the Template Browser plugin.
-->
<templates>
  <category name="Documents">
    <template name="Article" value="%3C%3Ch1%3EArticle+Title%3C%2Fh1%3E++%3Cdiv+style%3D%22+width%3A+30%25%
3B+float%3A+right%3B%22%3EInsert+image+here.%3C%2Fdiv%3E++%3Cp%3EInsert+article+text+here.%3C%2Fp%3E"/>
    <template name="Job Posting" value="%3C%3Ch1%3EJob+Title%3C%2Fh1%3E++%3Cp%
3EBrief+description+of+the+job.%3C%2Fp%3E++%3Ch2%3EResponsibilities%3C%2Fh2%3E++%3Cp%3EPrimary+responsibilities%
3A%3C%2Fp%3E++%3Cul%3E++%3C%3EList+of+the+key+responsibilities+for+the+job.%3C%2Fli%3E++%3C%2Ful%3E++%3Ch2%
3EEExperience%3C%2Fh2%3E++%3Cp%3EThe+desired+candidate+will+have%3A%3C%2Fp%3E++%3Cul%3E++%3C%3E
3EList+of+the+key+items+of+experience.%3C%2Fli%3E++%3C%2Ful%3E" />
    <template name="Press Release" value="%3Cp+style%3D%22text-align%3A+center%3B%22%3E%
5BInsert+company+logo+here%5D%3C%2Fp%3E++%3Ctable+style%3D%22+width%3A+100%25%3B+border-collapse%3A+collapse%3B%
22+cellpadding%3D%220%22+border%3D%220%22+cellspacing%3D%220%22%3E++%3Ctr%3E++%3Ctd+style%3D%22+width%3A+50%25%
3B+vertical-align%3A+top%3B%22%3EContact%3A+%3Cbr+%2F%3E+Tel%3A+%3Cbr+%2F%3E+Email%3A+%3C%2Ftd%3E++%3Ctd+style%
3D%22+text-align%3A+right%3B+width%3A+50%25%3B+vertical-align%3A+top%3B%22%3EFOR+IMMEDIATE+RELEASE%3C%2Ftd%3E++%
3C%2Ftr%3E++%3C%2Ftable%3E++%3Ch2+style%3D%22+text-align%3A+center%3B%22%3EMAIN+TITLE+OF+PRESS+RELEASE%3C%2Fh2%
3E++%3Ch4+style%3D%22+text-align%3A+center%3B%22%3ESubtitle+of+Press+Release%3C%2Fh4%3E++%3Cp+style%3D%22+text-
align%3A+left%3B%22%3EInsert+body+of+press+release+here.%3C%2Fp%3E" />
    <template name="Product Landing Page" value="%3Cdiv+style%3D%22+text-align%3A+center%3B+width%
3A+200%3B+height%3A+150%3B+padding%3A+4px+4px+0+4px%3B+border%3A+1px+solid+%23D4D4D4%3B+float%3A+left%3B%22%3E++%
3Cp%3E%5BInsert+image+here.%5D%3C%2Fp%3E++%3C%2Fdiv%3E++%3Cdiv+style%3D%22+text-align%3A+center%3B+height%3A+150%
3B%22%3E++%3Ch1%3EProduct+Name%3C%2Fh1%3E++%3Cp%3EShort+description+of+product%3C%2Fp%3E++%3C%2Fdiv%3E++%3Cp%3E%
26%23160%3B%3C%2Fp%3E++%3Cdiv+style%3D%22+background-color%3A+%23f8f8f8%3B+padding%3A+5px%3B%22%3E++%3Ch3%
3EProduct+Details%3C%2Fh3%3E++%3Cul%3E++%3C%3EProduct+Benefit+1%3C%2Fli%3E++%3C%3EProduct+Benefit+2%3C%2Fli%
3E++%3C%3EProduct+Benefit+3%3C%2Fli%3E++%3C%2Ful%3E++%3Ch3%3EFeature+Comparison%3C%2Fh3%3E++%
3Ctable+cellpadding%3D%220%22+border%3D%221%22+style%3D%22+width%3A+90%25%3B%22+cellspacing%3D%220%22%3E++%
3Ctr+style%3D%22+background-color%3A+%23C7D1DE%3B%22%3E++%3Ctd+style%3D%22+width%3A+14%25%3B+background-color%

```



```

20%3E%20%26%23160%3B%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3Cth%20style%3D%22color%3A%20white%3B%20%20border%
3A%201px%20solid%20%23D4D4D4%3B%20%20background-color%3A%20%23244872%3B%22%3E%20%26%23160%3B%3C%2Fth%3E%0A%3Ctd%
20style%3D%22border%3A%201px%20solid%20%23000024%3B%22%20%3E%20%26%23160%3B%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3Cth%
22border%3A%201px%20solid%20%23000024%3B%22%20%3E%20%26%23160%3B%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3Cth%
20style%3D%22color%3A%20white%3B%20%20border%3A%201px%20solid%20%23D4D4D4%3B%20%20background-color%3A%20%
23244872%3B%22%3E%20%26%23160%3B%3C%2Fth%3E%0A%3Ctd%20style%3D%22border%3A%201px%20solid%20%23000024%3B%22%20%
3E%20%26%23160%3B%3C%2Ftd%3E%0A%3Ctd%20style%3D%22border%3A%201px%20solid%20%23000024%3B%22%20%3E%20%26%23160%
3B%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3C%2Ftable%3E"/>
    </category>
  </category>
</templates>
<!--
Specify what accessibility checks are run in EditLive!
-->
<accessibilityChecks
  errors="true"
  warnings="true"
  manual="true"
  WCAG1="true"
  WCAG2="true"
  Section508="true"
  inlineAccessibility="false"
  emptyImageAlt="error"
  tableMappingIssues="warn"
/>
<mediaSettings>
  <!--
Specify HTTP upload settings
  'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
  'href' is your server-side script for handling multipart-formdata uploads from ELJ
  The httpUploadData element specifies any additional fields to post with the image data
  -->
  <!--
<httpUpload
  base="http://www.yourserver.com/userfiles/"
  href="http://www.yourserver.com/scripts/upload.jsp">
  <httpUploadData name="hello" data="world"/>
</httpUpload>
  -->
  <images allowLocalImages="true" allowUserSpecified="true" preferredWidth="800" preferredHeight="600">
  <!--
The list of images which appear in the Insert Image dialog.
TIP: Dynamically generate this from your database or repository to achieve an easy image library.
  -->
  <imageDialog width="700" height="350" />
  <imageList>
    <image name="Business team"
      alt="Business team meeting"
      border="0"
      title="Business team meeting"
      description="Photo of a business team meeting"
      src="http://static.ephox.com/demoimages/photos/TeamMeeting.jpg" />
    <image name="Note taking"
      alt="Photo of taking notes in a meeting"
      border="0"
      title="Photo of taking notes in a meeting"
      description="Photo of taking notes in a meeting"
      src="http://static.ephox.com/demoimages/photos/TakingNotes.jpg" />
    <image name="Businessman"
      alt="Photo of a businessman"
      border="0"
      title="Photo of a businessman"
      description="Photo of a businessman"
      src="http://static.ephox.com/demoimages/photos/businessman.jpg" />
    <image name="Bird"
      alt="Photo of a bird in a tree"
      border="0"
      title="Photo of a bird in a tree"
      description="Photo of a bird in a tree"
      src="http://static.ephox.com/demoimages/photos/bird.jpg" />
    <image name="Forest"

```

```

        alt="Photo of a New Zealand forest"
        border="0"
        title="Photo of a New Zealand forest"
        description="Photo of a forest"
        src="http://static.ephox.com/demoimages/photos/forest.jpg" />
<image name="Mountain"
    alt="Photo of a mountain"
    border="0"
    title="Photo of a mountain"
    description="Photo of a mountain"
    src="http://static.ephox.com/demoimages/photos/mountain.jpg" />
<image name="Penguin"
    alt="Yellow eyed penguin"
    border="0"
    title="Yellow eyed penguin"
    description="Yellow eyed penguin"
    src="http://static.ephox.com/demoimages/photos/penguin.jpg" />
<image name="Laptop"
    description="Laptop and mouse"
    alt="Person using a laptop"
    src="http://static.ephox.com/demoimages/photos/computerandmouse.jpg"
    title="Person using a laptop" />
<image name="Globe"
    alt="Globe icon"
    border="0"
    title="Globe icon"
    description="Globe icon"
    src="http://static.ephox.com/demoimages/icons/environment.png" />
<image name="Font"
    alt="Font icon"
    border="0"
    title="Font icon"
    description="Font icon"
    src="http://static.ephox.com/demoimages/icons/font.png" />
<image name="Lightbulb"
    alt="Lightbulb icon"
    border="0"
    title="Lightbulb icon"
    description="Lightbulb icon"
    src="http://static.ephox.com/demoimages/icons/lightbulb_on.png" />
<image name="Link"
    alt="Link icon"
    border="0"
    title="Link icon"
    description="Link icon"
    src="http://static.ephox.com/demoimages/icons/link.png" />
<image name="Star"
    alt="Green star icon"
    border="0"
    title="Green star icon"
    description="Green star icon"
    src="http://static.ephox.com/demoimages/icons/star_green.png" />
<image name="Wrench"
    alt="Wrench icon"
    border="0"
    title="Wrench icon"
    description="Wrench icon"
    src="http://static.ephox.com/demoimages/icons/wrench.png" />
</imageList>
</images>
<multimedia>
    <types>
        <type name="Adobe Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
            <param name="movie" />
            <param name="quality" />
            <param name="bgcolor" />
        </type>
        <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
            <param name="autohref" />
            <param name="autoplay" />

```



```

        <param name="bgcolor" />
        <param name="cache" />
        <param name="controller" />
        <param name="correction" />
        <param name="dontflattenwhensaving" />
        <param name="enablejavascript" />
        <param name="endtime" />
        <param name="fov" />
        <param name="height" />
        <param name="href" />
        <param name="kioskmode" />
        <param name="loop" />
        <param name="movieid" />
        <param name="moviename" />
        <param name="node" />
        <param name="pan" />
        <param name="playeveryframe" />
        <param name="qtsrcchokespeed" />
        <param name="scale" />
        <param name="starttime" />
        <param name="target" />
        <param name="targetcache" />
        <param name="tilt" />
        <param name="urlsubstitute" />
        <param name="volume" />
    </type>
    <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
        <param name="animationAtStart" />
        <param name="autoStart" />
        <param name="showControls" />
        <param name="clickToPlay" />
        <param name="transparentAtStart" />
    </type>
    <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
        <param name="animationAtStart" />
        <param name="autoStart" />
        <param name="showControls" />
        <param name="clickToPlay" />
        <param name="transparentAtStart" />
    </type>
    <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
    <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
    <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
</types>
</multimedia>
</mediaSettings>

<hyperlinks>
    <hyperlinkList>
        <hyperlink href="http://liveworks.ephox.com/documentation/editlive/v60/index.php?pageURL=RefHTML
/hyperlinklist.htm" description="How To Update This List" />
        <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
        <hyperlink href="http://liveworks.ephox.com" description="Ephox Developer Resources" />
        <hyperlink href="http://liveworks.ephox.com/support/" description="Ephox Support" />
        <hyperlink href="http://releases.ephox.com" description="Ephox Releases" />
        <hyperlink href="http://people.ephox.com" description="People@Ephox" />
    </hyperlinkList>
    <mailtoList>
        <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
        <mailtoLink href="mailto:sales@ephox.com" description="Ephox Sales" />
    </mailtoList>
</hyperlinks>
<!--
Customize the EditLive! menus
Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

```

```

<menu name="ephox_filemenu">
  <menuItem name="New" />
  <menuItem name="Open" />
  <menuSeparator />
  <menuItem name="Save" />
  <menuItem name="SaveAs" />
  <menuItem name="RestoreAutosave" />
  <menuSeparator />
  <menuItem name="ImportWordDocument" />
  <menuSeparator />
  <menuItem name="Print" />
</menu>
<menu name="ephox_editmenu">
  <menuItem name="Undo" />
  <menuItem name="Redo" />
  <menuSeparator />
  <menuItem name="Cut" />
  <menuItem name="Copy" />
  <menuItem name="Paste" />
  <menuItem name="PasteSpecial" />
  <menuSeparator />
  <menuItem name="Select" />
  <menuItem name="SelectAll" />
  <menuSeparator />
  <menuItem name="Find" />
  <menuSeparator />
</menu>
<menu name="ephox_viewmenu">
  <menuItemGroup name="SourceView" />
  <menuSeparator />
  <menuItem name="Popout" />
  <menuSeparator />
  <menuItem name="showDocumentNavigator" />
  <menuSeparator />
  <menuItem name="ParagraphMarker" />
</menu>
<menu name="ephox_insertmenu">
  <menuItem name="HLink" />
  <menuItem name="RemoveHyperlink" />
  <menuItem name="Bookmark" />
  <menuItem name="RemoveBookmark" />
  <menuSeparator />
  <menuItem name="ImageServer" />
  <menuItem name="InsertObject" />
  <menuItem name="InsertHTML" />
  <menuItem name="InsertTemplate" />
  <menuItem name="insertequation" />
  <menuSeparator />
  <menuItem name="CreateSection" />
  <menuItem name="RemoveSection" />
  <menuSeparator />
  <menuItem name="Symbol" />
  <menuItem name="HRule" />
  <menuSeparator />
  <menuItem name="DateTime" />
  <menuSeparator />
  <menuItem name="insertcomment" />
</menu>
<menu name="ephox_formatmenu">
  <submenu name="Style" />
  <submenu name="Face" />
  <submenu name="Size" />
  <menuSeparator />
  <menuItem name="Bold" />
  <menuItem name="Italic" />
  <menuItem name="Underline" />
  <menuSeparator />
  <menuItemGroup name="Align" />
  <menuSeparator />
  <menuItemGroup name="List" />
  <menuItem name="DecreaseIndent" />

```

```

    <menuItem name="IncreaseIndent" />
    <menuItem name="PropList" />
    <menuSeparator />
    <menuItem name="Color" />
    <menuItem name="HighlightColor" />
    <menuSeparator />
    <menuItemGroup name="Script" />
    <menuItem name="Strike" />
    <menuSeparator />
    <menuItemGroup name="textdirection" />
    <menuSeparator />
    <menuItem name="RemoveFormatting" />
    <menuItem name="FormatPainter" />
</menu>
<menu name="ephox_toolsmenu">
    <menuItem name="Spelling" />
    <menuItem name="BackgroundSpellChecking" />
    <menuItem name="Autocorrect" />
    <menuItem name="Thesaurus" />
    <menuSeparator />
    <!-- Enterprise Edition Feature -->
    <menuItem name="BrokenHyperlinkReport" />
    <menuSeparator />
    <!-- Enterprise Edition Feature -->
    <menuItem name="AccessibilityAsYouType" />
    <menuItem name="Accessibility" />
    <menuSeparator />
    <menuItem name="WordCount" />
</menu>
<!-- Enterprise Edition Features -->
<menu name="ephox_trackchangesmenu">
    <MenuItem name="AddComment" />
    <MenuItem name="RemoveAllComments" />
    <menuItem name="enabletrackchanges" />
    <menuSeparator />
    <menuItem name="acceptChange" />
    <menuItem name="rejectChange" />
    <menuSeparator />
    <menuItem name="previousChange" />
    <menuItem name="nextChange" />
    <menuSeparator />
    <menuItem name="acceptAllChanges" />
    <menuItem name="rejectAllChanges" />
    <menuSeparator />
    <menuItem name="showTrackChangesDialog" />
    <menuSeparator />
    <menuItem name="setUsername" />
</menu>
<menu name="ephox_tablemenu">
    <menuItem name="InsTable" />
    <menuItem name="InsRowCol" />
    <menuSeparator />
    <menuItem name="DelRow" />
    <menuItem name="DelCol" />
    <menuItem name="DelTable" />
    <menuSeparator />
    <menuItem name="Split" />
    <menuItem name="Merge" />
    <menuItem name="tableautofit" />
    <menuSeparator />
    <menuItem name="PropCell" />
    <menuItem name="PropRow" />
    <menuItem name="PropCol" />
    <menuItem name="PropTable" />
    <menuSeparator />
    <menuItem name="Gridlines" />
</menu>
<menu name="ephox_formmenu">
    <menuItem name="InsForm" />
    <menuSeparator />
    <menuItem name="InsTextField" />

```

```

    <menuItem name="InsPasswordField"/>
    <menuItem name="InsHiddenField"/>
    <menuItem name="InsFileField"/>
    <menuItem name="InsButtonField"/>
    <menuItem name="InsSubmitField"/>
    <menuItem name="InsResetField"/>
    <menuItem name="InsCheckboxField"/>
    <menuItem name="InsRadioField"/>
    <menuItem name="InsTextAreaField"/>
    <menuItem name="InsSelectField"/>
    <menuItem name="InsImageField"/>
  </menu>
  <menu name="ephox_help">
    <menuItem name="showHelp"/>
    <menuItem name="eljAboutELJ"/>
    <menuItem name="enableDebug"/>
  </menu>
</menuBar>

```

```
<!--
```

```
Customize the EditLive! toolbars
```

```
-->
```

```
<toolbars>
```

```

  <toolbar name="Command">
    <toolbarButton name="Print"/>
    <toolbarSeparator/>
    <toolbarButton name="Spelling"/>
    <toolbarButton name="Find"/>
    <toolbarSeparator/>
    <toolbarButton name="Cut"/>
    <toolbarButton name="Copy"/>
    <toolbarButton name="Paste"/>
    <toolbarButton name="FormatPainter" />
    <toolbarSeparator/>
    <toolbarButton name="Undo"/>
    <toolbarButton name="Redo"/>
    <toolbarSeparator/>
    <toolbarButton name="HLink"/>
    <toolbarButton name="ImageServer"/>
    <toolbarButton name="InsTableWizard"/>
    <toolbarSeparator/>
    <!-- Enterprise Edition Features -->
    <toolbarButton name="AccessibilityAsYouType" />
    <toolbarButton name="BrokenHyperlinkReport"/>
    <toolbarSeparator/>
    <!-- Enterprise Edition Features -->
    <toolbarButton name="AddComment"/>
    <toolbarButton name="enableTrackChanges"/>
    <toolbarButton name="acceptChange"/>
    <toolbarButton name="rejectchange"/>
    <toolbarButton name="previouschange"/>
    <toolbarButton name="nextchange"/>
    <toolbarSeparator/>
    <toolbarButton name="Popout"/>
  </toolbar>

```

```
<toolbar name="Format">
```

```
<!--
```

```
Styles from any embedded or external stylesheets will also be automatically added to the Styles
```

drop-down

```
-->
```

```

  <toolbarComboBox name="Style">
    <comboBoxItem name="P"/>
    <comboBoxItem name="H1"/>
    <comboBoxItem name="H2"/>
    <comboBoxItem name="H3"/>
    <comboBoxItem name="H4"/>
    <comboBoxItem name="H5"/>
    <comboBoxItem name="H6"/>
  </toolbarComboBox>

```

```
<!--
```

```
You can remove the Font drop-down if you just want users to use Styles.
```

```
The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
```

and Windows

To change the default font, change the embedded style sheet in the 'style' element above.

```
-->
<toolbarComboBox name="Face">
  <comboBoxItem name="Arial" text="Arial"/>
  <comboBoxItem name="Arial Black" text="Arial Black"/>
  <comboBoxItem name="Arial Narrow" text="Arial Narrow"/>
  <comboBoxItem name="Comic Sans MS" text="Comic Sans MS"/>
  <comboBoxItem name="Courier New" text="Courier New"/>
  <comboBoxItem name="Georgia" text="Georgia"/>
  <comboBoxItem name="Impact" text="Impact"/>
  <comboBoxItem name="Times New Roman" text="Times New Roman"/>
  <comboBoxItem name="Trebuchet MS" text="Trebuchet MS"/>
  <comboBoxItem name="Verdana" text="Verdana"/>
</toolbarComboBox>
<!--
Font Size drop-down
-->
<toolbarComboBox name="Size">
  <comboBoxItem name="1" text="7pt"/>
  <comboBoxItem name="2" text="8pt"/>
  <comboBoxItem name="3" text="10pt"/>
  <comboBoxItem name="4" text="12pt"/>
  <comboBoxItem name="5" text="14pt"/>
  <comboBoxItem name="6" text="18pt"/>
  <comboBoxItem name="7" text="24pt"/>
</toolbarComboBox>
<toolbarSeparator/>
<toolbarButton name="Bold"/>
<toolbarButton name="Italic"/>
<toolbarButton name="Underline"/>
<toolbarSeparator/>
<toolbarButtonGroup name="Align"/>
<toolbarSeparator/>
<toolbarButtonGroup name="List"/>
<toolbarButton name="DecreaseIndent"/>
<toolbarButton name="IncreaseIndent"/>
<toolbarSeparator/>
<toolbarButton name="HighlightColor"/>
<toolbarButton name="Color"/>
</toolbar>
</toolbars>
<!-- Note: Inline toolbars are currently only supported on img and table elements. -->
<inlineToolbars>
  <!-- Inline Image toolbar is an Enterprise Edition Feature -->
  <inlineToolbar name="img">
    <toolbarButton name="rotateCCW"/>
    <toolbarButton name="rotateCW"/>
    <toolbarSeparator />
    <toolbarButton name="flipVertical"/>
    <toolbarButton name="flipHorizontal"/>
    <toolbarSeparator />
    <toolbarButton name="reflect"/>
    <toolbarButton name="dropShadow"/>
    <toolbarButton name="roundedCorners"/>
    <toolbarSeparator />
    <toolbarButton name="crop"/>
  </inlineToolbar>
  <inlineToolbar name="table">
    <toolbarButton name="InsRow"/>
    <toolbarButton name="InsCol"/>
    <toolbarButton name="DelRow"/>
    <toolbarButton name="DelCol"/>
    <toolbarButton name="DelTable"/>
    <toolbarSeparator />
    <toolbarButton name="Split"/>
    <toolbarButton name="Merge"/>
    <toolbarSeparator />
    <toolbarButton name="tableautofit"/>
    <toolbarButton name="percentageTableSizing" />
    <toolbarButton name="pixelTableSizing" />
  </inlineToolbar>
</inlineToolbars>
```

```

        <!-- Enterprise Edition Features -->
        <toolbarSeparator />
        <toolbarButton name="ApplyCellHeaders"/>
        <toolbarButton name="ClearCellHeaders"/>
        <toolbarButton name="TableHeaderMappings"/>
        <toolbarSeparator />
        <toolbarButton name="Gridlines"/>
    </inlineToolbar>
</inlineToolbars>
<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo"/>
        <shrtMenuItem name="Redo"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Cut"/>
        <shrtMenuItem name="Copy"/>
        <shrtMenuItem name="Paste"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Select"/>
        <shrtMenuSeparator/>
        <!-- Enterprise Edition Features -->
        <shrtMenuItem name="AddComment"/>
        <shrtMenuItem name="acceptChange"/>
        <shrtMenuItem name="rejectChange"/>
        <shrtMenuItem name="nextchange"/>
        <shrtMenuItem name="previouschange"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Hyperlink"/>
        <shrtMenuItem name="RemoveHyperlink"/>
        <shrtMenuItem name="PropImage"/>
        <shrtMenuItem name="PropObject"/>
        <shrtMenuItem name="PropList"/>
        <shrtMenuItem name="PropHR"/>
        <shrtMenuItem name="PropSection"/>
        <shrtMenuItem name="PropForm"/>
        <shrtMenuItem name="PropFormField"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Split"/>
        <shrtMenuItem name="Merge"/>
        <shrtMenuItem name="tableautofit"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="PropTable"/>
        <shrtMenuItem name="PropRow"/>
        <shrtMenuItem name="PropCol"/>
        <shrtMenuItem name="PropCell"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="synonyms"/>
        <shrtMenuItem name="EditTag"/>
    </shrtMenu>
</shortcutMenu>
</editlive>

```

## Creating and Editing Configuration Files in Server Side Languages

Server side languages can be used to dynamically generate configuration files at runtime. This can be useful for situations such as displaying specific menu or toolbar items based on the user.

Configuration files can be stored in a variety of server side file types such as ASP, JSP, PHP and ColdFusion. When creating a configuration file to be used with a server side language, the file must contain the correct information to ensure rendering as XML.

### Example

For a configuration file *editliveconfig.jsp*, using Java server side processing, the following code can be used to specify rendering as XML.

```
<%@page contentType="text/xml"%>
```

Consult the documentation for your server side language on how to achieve this.

### Example

The following example demonstrates displaying the insert images toolbar item only if the user is identified as administrator.

```
<%@page contentType="text/xml"%>
<%
    //Get the user type information from the query string
    String userType = request.getParameter("user");
%>
<?xml version="1.0"?>
<editlive>
    <document>

    </document>
    ...
    <toolbars>
        <toolbar name="Command">
            ...
            <%
                //Allow administrator users to insert images
                if(userType.equals("administrator"))
                {
            %>
                <toolbarButton name="ImageServer"/>
            <%
                }
            %>
            ...
        </toolbar>
    </toolbars>
</editlive>
```

### See Also

- [EditLive! for Java Configuration File Elements](#)
- [Menu and Toolbar Item List](#)

# Licensing EditLive!

This article details how to license EditLive! for use within your application. Tiny products can be licensed in multiple ways; this article outlines how to make use of development and trial licenses, and how to request and install new licenses. Licenses issued by Tiny for EditLive! are bound by a licensing agreement outlined in the *license.txt* file available in the EditLive! SDK. All licenses are issued at the discretion of Tiny. For more information on licensing, please contact [Tiny](#).

## Development and Trial Licenses

The EditLive! SDK contains a license for the *LOCALHOST* domain which can be used for development purposes. In order to make use of the *LOCALHOST* development license supplied with EditLive!, the EditLive! applet must be accessed on the *LOCALHOST* domain. This license is neither time limited nor is it limited in the number of times it can be activated. Should a development license be required for a domain other than *LOCALHOST*, then please contact Tiny technical support.

EditLive! is also supplied with a 30-day trial license. This license is valid on any domain for 30 days from the date that EditLive! is first used on the client machine.

## Requesting a License

The licensing mechanism of the EditLive! applet is reliant on the domain from which the applet is accessed. When requesting a license from Tiny, please ensure that the domain name is correct for the system you are licensing EditLive! for. For the purposes of licensing EditLive!, the domain is considered to be everything between the trailing / of *http://* and the next /, excluding the port number. For example, if the EditLive! applet was to be accessed via the page *http://www.yourserver.com:8080/editor/editlive.html*, then the domain required for licensing would be *www.yourserver.com*.

Should EditLive! be required to be licensed on multiple subdomains, a license which enables use on those subdomains can be issued by Tiny on request. For instance, if the EditLive! applet is accessed from the URLs *http://www.yourserver.com/editor/editlive.html*, *http://intranet.yourserver.com/editor/editlive.html* and *http://yourserver.com/editor/editlive.html*, then a subdomain license for *yourserver.com* would be able to function as a license for these domains.

Licenses issued by Tiny for EditLive! are bound by a licensing agreement outlined in the *license.txt* file available in the EditLive! SDK. All licenses are issued at the discretion of Tiny.

## Installing a License

### Example

The following is an example EditLive! license. This is the evaluation license provided with EditLive! 8.0 for the *LOCALHOST* domain.

```
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-4DC5-EDF4-2486"
    licensee="For Evaluation Only"
    product="EditLive! for Java"
    release="8.0"
    type="Evaluation License"
    productivityPack="true"
  />
</ephoxLicenses>
```

The content of the `<license>` element in the *.lic* file must be added to the EditLive! configuration file in the `<ephoxLicenses>` element.

EditLive! license information can be added to a EditLive! configuration file by [Manually Editing Configuration Files](#).

### Example

The following shows an (abbreviated) EditLive! configuration file which contains a license. This example is taken from the EditLive! 6.0 sample configuration file.

```
<editLive>
  <document>
    ...
  </document>
  <ephoxLicenses>
    <license
      domain="LOCALHOST"
      key="6FFF-4DC5-EDF4-2486"
      licensee="For Evaluation Only"
      product="EditLive! for Java"
      release="8.0"
      type="Evaluation License"
      productivityPack="true"
    >
```



```
    />
  </ephoxLicenses>
  <htmlFilter ... />
  ...
</editLive>
```

## Installing Multiple Licenses

EditLive! can be used with multiple licenses. To use multiple licenses with EditLive!, each license should be specified in a distinct `<license>` element within the `<ephoxLicenses>` element of the EditLive! configuration file. EditLive! will attempt to register with each license listed in the configuration file in the order which they appear. All valid licenses listed will be activated; therefore, multiple limited seat licenses should not be used in the same configuration file.

## Timed Licenses

Upon request Tiny can issue a temporary, timed license for a specific domain for development purposes. These licenses include the expiry date in the domain name. For example, if the domain name for the timed license was *WWW.YOURSERVER.COM* and the license expired on October 30th, 2006, then the domain for the license key would appear as *WWW.YOURSERVER.COM20061030*. It is important that the date is present in the domain. Do *not* remove the eight (8) numbers on the end of the domain representing the expiry date or the license will not function.

## EditLive! Functionality Restricted by License Type

Various features of EditLive! can only be utilized by end users if a specific license type has been installed. The following functions can only be implemented in your instance of the editor if an [Enterprise Edition](#) license has been purchased (or if users are still operating under their 30 day trial period):

- [Track Changes](#)
- [Commenting](#)
- [Equation Editor](#)
- [Image Editing](#)
- [Advanced APIs](#)

## See Also

- [<ephoxLicenses> Configuration File Element](#)
- [<license> Configuration File Element](#)
- [Manually Editing Configuration Files](#)

# Autosave

Autosave will automatically save draft versions of content to the local machine before the content is saved to your content management system. Autosaves will occur every two minutes after the content has begun to be edited, preventing significant changes from being lost if the browser crashes or connectivity to the server is lost.

EditLive! will prompt the user on start-up if a draft for the current server exists that is newer than any known saved good version for the same server. Users can restore this draft (or the latest draft from any editing session in the past 24 hours) using either the notification bar displayed in the editor or a menu item.

Only one autosave will be kept for an editing session (a period of time where the editor is editing a specific piece of content), and these will be removed after 24 hours.

## Enabling Autosave

There are two steps to perform to enable Autosave.

### Enabling the Autosave Plugin

Autosave is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *autoSave* for a [<plugin>](#) element resident in your EditLive! [Configuration File](#).

#### Example

```
<editlive>
  ...
  <plugins>
    <plugin name="autosave" />
  </plugins>
</editlive>
```

### Specifying the Autosave Toolbar and/or Menu Items

The [Menu and Toolbar Item List](#) features a toolbar/menu item for displaying the list of Autosave content items available to restore from.

## Notification bar

The notification bar will be displayed by default if there is a draft available. Displaying the notification bar can be controlled using the *showinfo* attribute for the Autosave plugin (from EditLive! 9.1.0.230).

```
<plugin name="autosave" showinfo="false" />
```

# Automatic Hyperlinking

The Automatic Hyperlinking feature of EditLive! automatically converts URLs into clickable hyperlinks.

## Enabling Autolink

Automatic Hyperlink is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *autolink* for a `<plugin>` element resident in your EditLive! [Configuration File](#) to enable Automatic Hyperlink functionality.

### Example

```
<editlive>
  ...
  <plugins>
    <plugin name="autolink" />
  </plugins>
</editlive>
```

# Insert HTML Fragment

The Insert HTML Fragment feature of EditLive! is a simple dialog where users can paste HTML which will be then inserted directly into the document.

## Enabling Insert HTML Fragment

There are two steps to perform in order to enable the Insert HTML Fragment feature.

### Enabling the Insert HTML Fragment Plugin

Insert HTML Fragment is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *insertHTML* for a [<plugin>](#) element resident in your EditLive! [Configuration File](#).

#### Example

```
<editlive>
  ...
  <plugins>
    <plugin name="insertHTML" />
  </plugins>
</editlive>
```

### Specifying the Commenting Menu Items

The [Menu and Toolbar Item List](#) features a menu/toolbar item for Insert HTML Fragment that will display the dialog into which users can paste HTML.

# Java Webstart for Java 9+

In Java 9+ the core parts of Java were split into modules and accessing them required new permissions. As EditLive! was written before the module system it requires quite a few permissions to run as shown in this sample JNLP file.

## JNLP sample

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://host/path/to/jnlp/" href="yourapp.jnlp">
  <information>
    <title>Jnlp Example</title>
    <vendor>Company Name</vendor>
    <homepage href="http://example.com" />
    <description>Example of a JNLP file for EditLive!</description>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <java version="1.8+" java-vm-args="--add-exports=java.desktop/sun.swing=ALL-UNNAMED --add-exports=java.
desktop/sun.swing.table=ALL-UNNAMED --add-exports=java.desktop/sun.swing.plaf.synth=ALL-UNNAMED --add-
opens=java.desktop/javafx.swing.plaf.synth=ALL-UNNAMED --add-opens=java.desktop/javafx.swing.plaf.basic=ALL-
UNNAMED --add-opens=java.desktop/javafx.swing=ALL-UNNAMED --add-opens=java.desktop/javafx.swing.tree=ALL-UNNAMED
--add-opens=java.desktop/java.awt.event=ALL-UNNAMED --add-exports=java.desktop/com.sun.java.swing.plaf.
windows=ALL-UNNAMED --add-exports=java.desktop/sun.awt.shell=ALL-UNNAMED --add-exports=java.desktop/com.sun.
awt=ALL-UNNAMED --add-exports=java.base/sun.security.action=ALL-UNNAMED --add-opens=java.base/sun.util.
calendar=ALL-UNNAMED --add-opens=java.desktop/javafx.swing.text=ALL-UNNAMED --add-opens=java.desktop/javafx.
swing.text.html=ALL-UNNAMED --add-opens=java.desktop/javafx.swing.text.html.parser=ALL-UNNAMED" />
    <!-- Note as normal for Java Web Start you must have signed all the jars with the same key -->
    <jar href="yourapp.jar" />
    <jar href="editlivejava.jar" />
  </resources>
  <application-desc main-class="YourMainClass" />
</jnlp>
```

# Enterprise Edition

- [Enabling Enterprise Edition](#)
- [Enterprise Edition Features](#)

# Enabling Enterprise Edition

The Enterprise Edition of EditLive! provides users with an array of exclusive new features for the editor. View the complete list of [Enterprise Edition Features](#).

## Enabling Enterprise Edition

Enterprise Edition is activated by installing an Enterprise Edition license for the editor provided by Tiny. EditLive! licenses with Enterprise Edition specified are installed in the same manner as regular EditLive! licenses. In order to obtain an Enterprise Edition license, contact Tiny via the [website](#).

For more information on how to install licenses for EditLive!, see the [Licensing EditLive!](#) article of this SDK.

The features available in the Enterprise Edition of the editor will also work for users still operating under their 30 day trial period.

# Enterprise Edition Features

## Track Changes

Track Changes provided users with an easy-to-use interface for both viewing changes to a document and accepting/rejecting individual changes.

- [Getting Started With Track Changes](#)
- [Menu and Toolbar Item List](#)
- [trackChanges](#) Configuration Option
- [setUserName](#) Method

## Commenting

The Commenting feature of EditLive! allows users to add comments to selections of text in a document. This feature integrates with the Track Changes feature by using the same usernames and colors to identify comments made within the document.

- [Commenting](#)
- [<plugin>](#) Configuration Element

## Image Editing

Image Editing allows users to perform various image manipulation techniques on images in the document.

- [Image Editing](#)
- [<images>](#) Configuration Element
- [<plugin>](#) Configuration Element

## Accessibility As You Type

Accessibility As You Type allows users to quickly identify and edit content that does not meet key web accessibility guidelines (such as the [W3C Accessibility Compliance Guidelines](#) and the [United States Section 508 Accessibility Guidelines](#)).

- [Accessibility As You Type](#)
- [<plugin>](#) Configuration Element

## Accessibility Reports

Accessibility Reports reveal areas of content that are not compliant with key web accessibility guidelines (such as the [W3C Accessibility Compliance Guidelines](#) and the [United States Section 508 Accessibility Guidelines](#)) and directs users to where correction is needed. The accessibility report will give a short explanation of the errors and a recommendation as to how it should be corrected.

- [Accessibility Compliance](#)
- [<plugin>](#) Configuration Element

## Table Accessibility

EditLive! features various table-specific functionalities which aid in the creation of accessible content.

- [Table Accessibility](#)
- [<plugin>](#) Configuration Element

Only *some* of the features available in Table Accessibility require an Enterprise Edition license. Please refer to the [Menu and Toolbar Item List](#) for more information.

## Broken Hyperlink Report

The Broken Hyperlink Report allows users to verify that the hyperlinks in the EditLive! document are valid. In order to enable the Broken Hyperlink Report, you will need to perform the following:

- Add a [<plugin>](#) element named *BrokenHyperlinkReport* to your configuration file.
- Add the [Broken Hyperlink Report](#) menu item to your configuration file.

## Template Browser

The Template Browser feature allows users to insert templates into any document they are editing with EditLive!. In order to enable the Template Browser, you will need to perform the following:

- Add a [<plugin>](#) element named *templateBrowser* to your configuration file.
- Add the [Template Browser](#) menu item to your configuration file. By default, this menu item will appear in the Insert menu.

### Adding Templates



Sample templates are packaged by default with EditLive!. Because of this, you do not need to specify any templates to be able to use the Template Browser. However, if you would like users to have custom templates, you will need to add them to the `<templates>` element resident in your EditLive! [Configuration File](#). Specifically, you will need to specify a name and a URL-encoded value attribute for each `<template>` element you wish to define as in the following example:

```
<editlive>
  ...
  <templates>
    <category name="Category 1">
      <template name="Template 1" value="Template%201" />
    </category>
  </templates>
</editlive>
```

## Equation Editing

Equation Editing provides users with an interface for creating and editing mathematical equations.

- [Integrating the Equation Editor](#)
- [setUseMathML Method](#)
- [Equation Editor Menu and Toolbar Items](#)
- `<mathml>` Configuration Element

## Advanced APIs

The Swing SDK APIs are available to applet users as Advanced APIs. They provide developers with the ability to interact with the editor using Java code.

- [Introduction to the Advanced APIs](#)
- [addJar Method](#)
- [Creating and Using Plugins in the Applet](#)
- [Plugin XML Elements](#)
- [addPlugin Method](#)
- [addPluginAsText Method](#)

# Editor Appearance

- [Setting Menu and Toolbar Items](#)
  - [Mnemonics and Shortcuts for Menus](#)
- [Menu and Toolbar Item List](#)
  - [File Commands](#)
  - [Edit Commands](#)
  - [View Commands](#)
  - [Insert Commands](#)
  - [Format Commands](#)
  - [Tool Commands](#)
  - [Table Commands](#)
  - [Properties Commands](#)
  - [Help Commands](#)
  - [Form Commands](#)
  - [Track Changes Commands](#)
  - [Image Editor Commands](#)
  - [Accessibility Commands](#)
  - [Broken Hyperlink Report](#)
  - [Commenting Commands](#)
  - [Equation Editor Commands](#)
  - [Menu and Toolbar Item Groups](#)
- [Creating Custom Menu and Toolbar Items](#)
- [Customizing the Color Picker](#)

# Setting Menu and Toolbar Items

The menus and toolbars of Tiny EditLive! are completely customizable through configuration files. This article explains how the EditLive! toolbars and menu bars may be configured and how the configuration of the menu bar and toolbars affects the functionality of EditLive!.

## The Menu Bar

The menu bar in EditLive! can have any number of individual menus added to it. The names of the menus added to the menu bar are completely customizable. Furthermore, the specification of a mnemonic for the menu is also customizable. To specify the mnemonic for a menu, an escaped ampersand (&) must be specified in the name attribute of the relevant `<menu>` element in the XML configuration file.

### Example

This would specify a View menu which has the mnemonic of V:

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="&View">
      ...
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

Tiny also provides 8 default menu names. These default menu names are automatically internationalized to match the user's locale.

- `ephox_filemenu` - The internationalized *File* menu.
- `ephox_editmenu` - The internationalized *Edit* menu.
- `ephox_viewmenu` - The internationalized *View* menu.
- `ephox_insertmenu` - The internationalized *Insert* menu.
- `ephox_formatmenu` - The internationalized *Format* menu.
- `ephox_toolsmenu` - The internationalized *Tools* menu.
- `ephox_tablemenu` - The internationalized *Table* menu.
- `ephox_formmenu` - The internationalized *Forms* menu.

### Example

The following configuration file example will display a View menu item in whatever native language the user's local machine is using.

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="ephox_viewmenu">
      ...
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

## Menu Items

EditLive! menu items are specified using the **name** attribute of a `<menuitem>` configuration file element. Any number of `<menuitem>` elements can be nested within a `<menu>` configuration file element. For a comprehensive list of all available menu items see the [Menu and Toolbar Item List](#).

### Example

The following configuration file example would create an Open menu item within a File menu.

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="ephox_filemenu">
      ...
    </menu>
  </menuBar>
  ...
</editLive>
```

```

        <menuItem name="Open" />
        ...
    </menu>
    ...
</menuBar>
...
</editLive>

```

## Menu Item Groups

Some menu items are added to the applet's interface in a group. These groups of menu items within EditLive! are specified through the use of a [<menuItemGroup>](#) configuration file element. The **name** attribute of the [<menuItemGroup>](#) element determines which menu item group is inserted. EditLive! comes with a collection of predefined interface item groups which may be placed in the menus and toolbars. Menu item groups will have default menu item text associated with them. They may also have default mnemonics, images, and shortcuts. For more information on what menu item groups are available for use, please see the [Menu and Toolbar Item List](#) article. The activation of menu items in menu item groups is mutually exclusive; for example, the Browser View item cannot be activated at the same time as the Window View item.

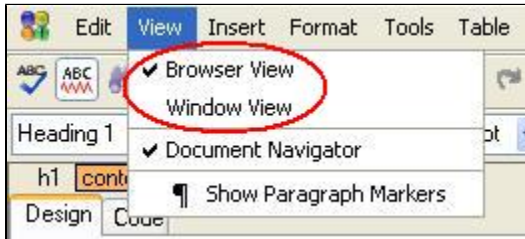
### Example

The following configuration file example would add the Browser View and Window View items to the View menu.

```

<editLive>
...
<menuBar>
...
  <menu name="ephox_viewmenu">
    ...
    <menuItemGroup name="FrameView" />
    ...
  </menu>
...
</menuBar>
...
</editLive>

```



## Menu Separators

Menu separators are horizontal lines spanning the width of the menu which can be used to visually break a menu into its constituent parts and areas. These are added through the use of the [<menuSeparator>](#) configuration file element, within a [<menu>](#) element. They serve no purpose other than that of a visual aid.

## Toolbars

The EditLive! applet can be instantiated with multiple toolbars. The [<toolbar>](#) element is used to specify a toolbar within the EditLive! configuration file. Toolbars will appear within EditLive in the order in which they are listed within the configuration file. When specifying a toolbar with the [<toolbar>](#) element, the toolbar must be given a unique name via the **name** attribute. The value of the **name** attribute does not appear in the EditLive! user interface.

### Example

The following configuration file example would specify a new toolbar with the name *format*:

```

<editLive>
...
<toolbars>
...
  <toolbar name="format">
    ...
  </toolbar>
...

```

```

</toolbars>
...
</editLive>

```

Each toolbar can have a variety of buttons, button groups, and drop-down combo boxes added to it. Toolbar buttons are added via the `<toolbarButton>` element, toolbar button groups through the `<toolbarButtonGroup>` element, and combo boxes through the `<toolbarComboBox>` element.

### Toolbar Buttons

Toolbar buttons in EditLive! are specified through the use of a `<toolbarButton>` configuration file element. The name attribute of the `<toolbarButton>` element determines which toolbar button is inserted. EditLive! comes with a collection of predefined interface items which may be placed on the EditLive! toolbars, menus, or shortcut menu. Items from the [Menu and Toolbar Item List](#) will have default tool tip text associated with them. Most also have default images; however, some of the items may not have images associated with them. It is recommended that interface commands without associated images are not placed on the toolbars. For more information on what interface commands are available for use please see the [Menu and Toolbar Item List](#) article.

#### Example

The following configuration file example would add the New, Open..., Save and Save As... items to a toolbar with the name *command*:

```

<editLive>
...
<toolbars>
...
<toolbar name="command">
  <toolbarButton name="New"/>
  <toolbarButton name="Open"/>
  <toolbarButton name="Save"/>
  <toolbarButton name="SaveAs"/>
</toolbar>
...
</toolbars>
...
</editLive>

```

### Toolbar Button Groups

Some toolbar buttons are added to the applet's interface in a group. These groups of toolbar buttons within EditLive! are specified through the use of a `<toolbarButtonGroup>` configuration file element. The **name** attribute of the `<toolbarButtonGroup>` element determines which toolbar button group is inserted. EditLive! comes with a collection of predefined interface item groups which may be placed on the menus and toolbars. Items from the interface command collection will have default tool tip text associated with them. Most also have default images; however, some of the items may not have images associated with them. It is recommended that interface item groups without associated images are not placed on the interface. For more information on what interface commands are available for use please see the EditLive! for Java [Menu and Toolbar Item List](#) article.

#### Example

The following configuration file example would add the Align Left, Align Center and Align Right buttons from the alignment button group to the toolbar named *command*.

```

<editLive>
...
<toolbars>
...
<toolbar name="command">
  <toolbarButtonGroup name="Align"/>
</toolbar>
...
</toolbars>
...
</editLive>

```



### Toolbar Combo Boxes

Combo boxes for the style, font typeface, and font size text attributes can be added to the EditLive! toolbar through the use of a `<toolbarComboBox>` configuration file element with a specific value for the name attribute. Each toolbar combo box has a specific value for the **name** attribute associated with it.

Furthermore, the items listed in each of these combo boxes can be specified by the developer through the inclusion of `<comboBoxItem>` child elements in a `<toolbarComboBox>` element.

### Example

The following configuration file example would create the Style drop down combo box in the EditLive! for Java Format Toolbar with the listing of Normal, Heading 1, Heading 2, and Heading 3 items which represent the `<P>`, `<H1>`, `<H2>`, and `<H3>` styles respectively:

```
<editLive>
...
<toolbars>
...
<toolbar name="format">
...
<toolbarComboBox name="Style">
  <comboBoxItem name="P" text="Normal" />
  <comboBoxItem name="H1" text="Heading 1" />
  <comboBoxItem name="H2" text="Heading 2" />
  <comboBoxItem name="H3" text="Heading 3" />
</toolbarComboBox>
...
</toolbar>
...
</toolbars>
...
</editLive>
```

The toolbar combo boxes available for use with EditLive!, their corresponding functions, and their associated values for the `<toolbarComboBox>` name attribute are listed in the [Toolbar Combo Box Items](#) section of the [Menu and Toolbar Item List](#).

### Toolbar Separators

Toolbar separators are vertical lines spanning the height of the toolbar that visually break a toolbar into its constituent parts and areas. These are added through the use of the `<toolbarSeparator>` configuration file element within a `<toolbar>` element. They serve no purpose other than that of a visual aid.

### Inline Toolbars

EditLive! provides support for a select set of toolbars which hover above specific HTML elements. Inline toolbars are specified via `<inlineToolbar>` configuration elements, which are nested under the `<inlineToolbars>` configuration element.

The **name** attribute for an `<inlineToolbar>` element specifies which element the toolbar will hover above. Consult the `<inlineToolbar>` documentation for a list of the currently supported **name** attribute values.

`<inlineToolbar>` elements support child `<toolbarButton>` and `<toolbarSeparator>` configuration elements to display inline toolbar buttons and separators.

### Example

The following configuration file example creates an inline toolbar that hovers over TABLE elements. This toolbar features the *Insert Table Row* and *Insert Table Column* toolbar buttons.

```
<editLive>
...
<inlineToolbars>
  <inlineToolbar name="table">
    <toolbarButton name="InsRow"/>
    <toolbarButton name="InsCol"/>
  </inlineToolbar>
</inlineToolbars>
...
</editLive>
```

### The Shortcut (Context) Menu

Shortcut menu items in EditLive! are specified through the use of a `<shrtMenuItem>` configuration file element, with a specific value for the **name** attribute associated with it. Any of the **name** attributes applicable to `<menuItem>` and `<toolbarButton>` configuration file elements can be used in the shortcut menu.

### Example

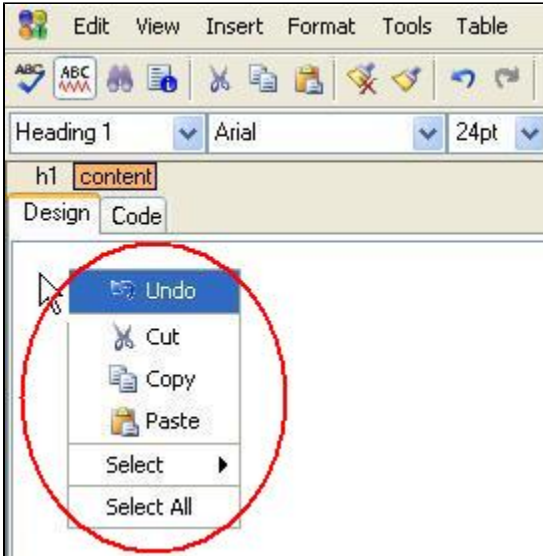
The following configuration file example would add the Cut, Copy and Paste items to the Shortcut menu:

```
<editLive>
...
<shortcutMenu>
```

```

    <shrtMenuItem name="Cut" />
    <shrtMenuItem name="Copy" />
    <shrtMenuItem name="Paste" />
  </shrtMenuItem>
</shrtMenuItem>
</shortcutMenu>
...
</editLive>

```



## Submenus

Submenus can be added to the EditLive! menu bar and shortcut menu. These are added through the use of the `<submenu>` configuration file element with a specific value for the **name** attribute.

If a `<submenu>` element is left empty, each of the submenu added will contain the same items as the corresponding item on the EditLive! toolbar. If the corresponding item does not exist on the toolbar then the submenu will appear empty. If the developer wishes to make the submenu items distinct from the toolbar items, the `<submenu>` element may have `<menuItem>` child elements added to it.

Once menu items are specified within a submenu then the contents of the submenu will no longer mirror the corresponding toolbar element.

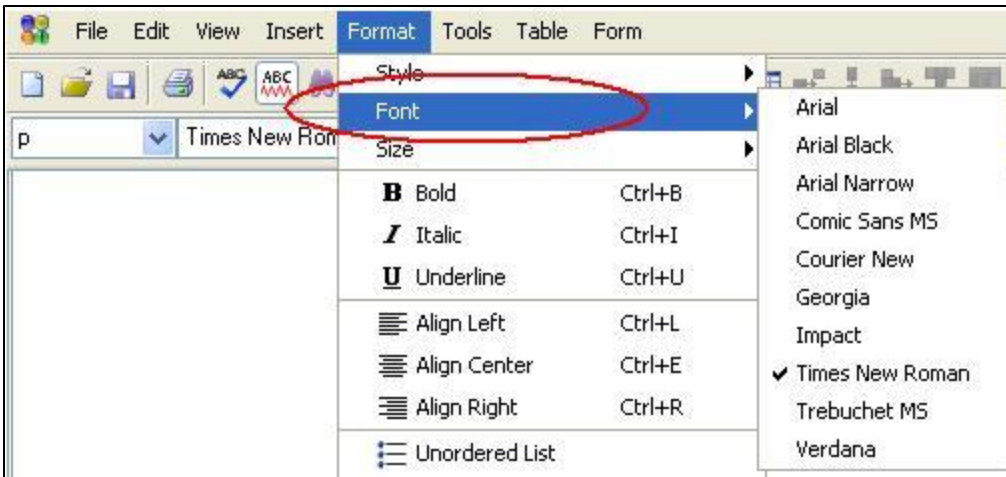
### Example

The following configuration file example would add the Font Face submenu to the Format menu with items on the submenu corresponding to the items specified in the Font Face drop down combo box of the EditLive! toolbar:

```

<editLive>
...
<menuBar>
...
  <menu name="ephox_formatmenu">
    ...
    <submenu name="Face" />
    ...
  </menu>
...
</menuBar>
...
</editLive>

```



**Example**

The following configuration file example would add the Style submenu to the Shortcut menu with items on the submenu corresponding to the items specified in the Font drop down combo box of the EditLive! toolbar:

```

<editLive>
...
<shortcutMenu>
  <shrtMenu>
    ...
    <submenu name="Style" />
    ...
  </shrtMenu>
</shortcutMenu>
...
</editLive>

```

**Example**

The following configuration file example would add the Style submenu to the Format menu. The menu created would contain the Normal and Heading 1 Styles which, respectively, correspond to the <P> and <H1> styles. Note that this submenu would **NOT** contain the values from the corresponding toolbar item.

```

<editLive>
...
<menuBar>
  ...
  <menu name="ephox_formatmenu">
    ...
    <submenu name="Style">
      <menuItem name="P" text="Normal" />
      <menuItem name="H1" text="Heading 1" />
    </submenu>
    ...
  </menu>
  ...
</menuBar>
...
</editLive>

```

**Example**

The following configuration file example would a customized Color submenu to the Format menu. The color submenu would contain the colors red, blue, and yellow.

```

<editLive>
...
<menuBar>
  ...
  <menu name="ephox_formatmenu">
    ...
    <submenu name="Color">

```



```

        <menuItem name="#FF0000" text="Red" />
        <menuItem name="#0000FF" text="Blue" />
        <menuItem name="#FFFF00" text="Yellow" />
    </submenu>
    ...
</menu>
...
</menuBar>
...
</editLive>

```

The submenus available for use with EditLive!, their corresponding toolbar items, mnemonics, and associated values for the <submenu> **name** attribute are listed in the [Menu and Toolbar Item List](#).

## Customizing Available Items

The interface items available through the standard EditLive! interface command collection can be customized to allow the associated text, mnemonic, and image to be altered. These customizations can be performed on menu and toolbar items by setting the **text**, **imageURL** and **mnemonic** attributes of the <menuItem> and <shrtMenuItem> configuration file elements, and the **text** and **imageURL** attributes of the <toolbarButton> element.

### Example

The following example customizes the New menu item to use the text Create New, the image *customImage.gif* and the mnemonic c.

```

<editLive>
...
<menuBar>
...
<menu name="&File">
    <menuItem name="New" text="Create New" imageURL="customImage.gif" mnemonic="c" />
    ...
</menu>
...
</menuBar>
...
</editLive>

```

## Tab Views

EditLive! can be configured to use a tabbed view which allows users to more intuitively switch between different views. These tabs can be placed on the top or bottom of the EditLive! editing area, or they can be removed completely. The configuration of the tabbed view for EditLive! is achieved via the **tabPlacement** attribute of the <wysiwygEditor> configuration file element.

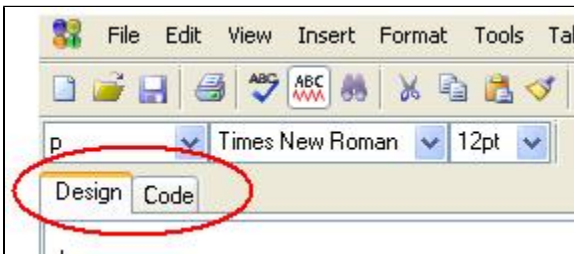
### Example

For the following configuration file example, the views tab would appear at the top of the EditLive! editing pane.

```

<editLive>
...
<wysiwygEditor tabPlacement="top">
...
</wysiwygEditor>
...
</editLive>

```



## Removing the Toolbars

Either of the toolbars of EditLive! may be removed. In order to display EditLive! without any toolbars, ensure that there are no <toolbar> elements within the configuration file.

## Limiting the Functionality of EditLive!

Removing specific functionality from EditLive! is achieved by removing the corresponding menu and/or toolbar buttons from the EditLive! interface. When the item is not included in the configuration file then the item will not appear on the menu or toolbar and, in most cases, the shortcut key for the item will be disabled.

The shortcut keys for the Cut, Copy, Paste, Bold, Italic and Underline actions will always be enabled. This is independent of the associated menu items and toolbar buttons. Thus, the shortcut keys for these functions will still be functional even if the associated menu items or toolbar buttons are removed.

## See Also

- [Configuration File](#)
- [Manually Editing Configuration Files](#)
- [Menu and Toolbar Item List](#)

# Mnemonics and Shortcuts for Menus

Using an EditLive! configuration file, you can create your own mnemonics and shortcuts for any menu or menu item appearing in EditLive!.

The following attributes can be applied to the following configuration file elements:

- `<menu>`
- `<menuItem>`
- `<customMenuItem>`

## mnemonic

A mnemonic allows developers to load specific menus and menu items by pressing combinations of ALT and a specific keyboard key. When holding ALT, each menu and menu item will display an underlined letter. Pressing the corresponding keyboard key will activate the menu or menu item.

Assign a single character to the mnemonic attribute.

### Example

The following custom menu item has a mnemonic set to the A key.

```
<customMenuItem
  name="customMenuItem1"
  text="Custom Menu Item"
  action="raiseEvent"
  value="customMenuItem1Raised"
  mnemonic="A" />
```

## shortcut

Shortcuts define an explicit combination of keyboard keys that will invoke the specified menu, menu item or custom menu item. The value assigned to the shortcut attribute is required to adhere to the [standard Java Keystroke class](#).

### Example

The following custom menu item will be invoked if the user presses the CTRL, ALT and X keys simultaneously.

```
<customMenuItem
  name="customMenuItem1"
  text="Custom Menu Item"
  action="raiseEvent"
  value="customMenuItem1Raised"
  shortcut="control alt X" />
```

### Example

The following custom menu item will be invoked if the user presses the SHIFT and the DELETE keys simultaneously.

```
<customMenuItem
  name="customMenuItem1"
  text="Custom Menu Item"
  action="raiseEvent"
  value="customMenuItem1Raised"
  shortcut="shift DELETE" />
```

## Remarks

Many `<menuItem>` configuration elements already have mnemonics and shortcuts specified. If you use the mnemonic or shortcut attributes on these elements, you will override the default.

For a list of the default mnemonics and shortcuts for the menus and menu items available for EditLive!, see the [Menu and Toolbar Item List](#) in the Developer Guide for this SDK.

# Menu and Toolbar Item List

Menu items and toolbar items can be easily added to an EditLive! configuration file.

If you are manually editing an EditLive! configuration file, the following commands can be used with any [<menuItem>](#), [<toolbarButton>](#) or [<shrtMenuItem>](#) element with the configuration file.

## Example

The menu item **Open** has the XML name attribute *Open*. To create an **Open** menu item, enter the following between `<menu></menu>` tags found in the configuration file.

```
<menu name="ExampleMenu" >
  <menuItem name="Open" />
</menu>
```

For more information on how to add Menu and Toolbar items, please see the [Setting Menu and Toolbar Items](#) section of this SDK.





## Menu and Toolbar Items

- [File Commands](#)
- [Edit Commands](#)
- [View Commands](#)
- [Insert Commands](#)
- [Format Commands](#)
- [Tool Commands](#)
- [Table Commands](#)
- [Properties Commands](#)
- [Help Commands](#)
- [Form Commands](#)
- [Track Changes Commands](#)
- [Image Editor Commands](#)
- [Accessibility Commands](#)
- [Broken Hyperlink Report](#)
- [Commenting Commands](#)
- [Equation Editor Commands](#)
- [Menu and Toolbar Item Groups](#)







## See Also

- [Setting Menu and Toolbar Items](#)
- [<menuItem>](#) Configuration File Element
- [<toolbarButton>](#) Configuration File Element

# File Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Create a new file.	New	New	CTRL+N		N
Open an existing file on the local machine.	Open	Open...	CTRL+O		O
Save a file to the local machine.	Save	Save	CTRL+S		S
Save a file to the local machine with a different name.	SaveAs	Save As...	CTRL+SHIFT+S	N/A	A
Display a dialog to restore an autosaved version of a piece of content	RestoreAutosave	Restore from Autosave...	N/A	N/A	N/A
Import a Word document.	ImportWordDocument	Import Word Document...	N/A	N/A	N/A
Print the contents of EditLive!	Print	Print..	N/A		P




# Edit Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Undo the last editor action.	Undo	Undo	CTRL+Z		U
Redo the last undone editor action.	Redo	Redo	CTRL+Y		R
Cut the selection.	Cut	Cut	CTRL+X		T
Copy the selection.	Copy	Copy	CTRL+C		C
Paste.	Paste	Paste	CTRL+V		P
Paste Special.	PasteSpecial	Paste Special	N/A	N/A	S
Select all editor content.	SelectAll	Select All	CTRL+A	N/A	L
Find text in the editor.	Find	Find...	CTRL+F		F















# View Commands

## View Commands

Some of the items listed below can be added individually to the editor as opposed to adding an entire group of items. To display individual items listed below, use the *XML Name Attribute (Individual Item)* where specified. XML groups cannot be added to the shortcut menu within EditLive!.

Function	XML Group Attribute	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
View the document in Design mode (WYSIWYG mode).	SourceView	DesignView	Design View	N/A	N/A	D
View and edit the HTML source for the document.	SourceView	HTMLView	Code View	N/A	N/A	H
Display EditLive! in a separate window (applet only)	N/A	Popout	Window View	N/A		W
Show or hide document navigator.	N/A	ShowDocumentNavigator	Document Navigator	N/A	N/A	N
Show or hide paragraph markers and editing grid lines.	N/A	ParagraphMarker	Show/Hide Paragraph Markers	N/A		P
View the document within a set page width. Choices include <ul style="list-style-type: none"> <li>• Smartphone Portrait</li> <li>• Smartphone Landscape</li> <li>• Tablet Portrait</li> <li>• Tablet Landscape</li> <li>• Monitor</li> <li>• Widescreen Monitor</li> <li>• Size to Fit</li> <li>• Custom</li> </ul>	N/A	pagewidth	Page Width	N/A		N/A

# Insert Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Insert a hyperlink.	HLink	Insert Hyperlink...	CTRL+K		H
Remove a hyperlink.	RemoveHyperlink	Remove Hyperlink	N/A		H
Insert a bookmark.	Bookmark	Bookmark...	N/A		K
Remove a bookmark.	RemoveBookmark	Remove Bookmark	N/A		K
Insert an image (local or server).	ImageServer	Insert Image...	N/A		I
Insert Media	InsertMedia	Insert Media...	N/A		I
Insert an embedded object or multimedia file.	InsertObject	Insert Object...	N/A	N/A	N/A
Insert a fragment of HTML.	InsertHTML	Insert HTML Fragment...	N/A		N/A
View and insert templates.	InsertTemplate	Insert Template...	N/A		N/A
Create a section around the selected text. Displays a menu of section types, including: <ul style="list-style-type: none"> <li>• BlockQuote</li> <li>• Div</li> <li>• Section</li> <li>• Header</li> <li>• Footer</li> <li>• Article</li> <li>• Aside</li> <li>• Nav</li> <li>• Figure</li> <li>• FigCaption</li> </ul>	CreateSection	Create Section	N/A		C
Remove a section around the selected text. Displays a menu of section types, e.g. div, blockquote, article	RemoveSection	Remove Section	N/A		N/A
Insert a symbol.	Symbol	Insert Symbol...	N/A		S
Insert a horizontal line.	HRule	Insert Horizontal Rule	N/A		L
Insert a date and time.	DateTime	Insert Date and Time...	N/A		T
Insert a HTML comment.	InsertComment	Insert HTML Comment...	N/A		N/A
Edit a HTML comment, scripting tag or unknown custom tag.	EditTag	Edit Tag...	N/A	N/A	N/A








# Format Commands

Some of the items listed below can be added individually to the editor as opposed to adding an entire group of items. To display individual items listed below, use the *XML Name Attribute (Individual Item)* where specified. XML groups cannot be added to the shortcut menu within EditLive!.

Function	XML Group Attribute	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic	Notes
Bold text.	N/A	Bold	Bold	CTRL+B		B	
Italic text.	N/A	Italic	Italic	CTRL+I		I	
Underline text.	N/A	Underline	Underline	CTRL+U		U	
Set left alignment.	Align	AlignLeft	Align Left	CTRL+L		L	
Set center alignment.	Align	AlignCenter	Align Center	CTRL+E		C	
Set right alignment.	Align	AlignRight	Align Right	CTRL+R		R	
Set justified alignment.	N/A	AlignJustify	Align Justified	CTRL+J		J	Added in EditLive 9.0.1.611  Note that EditLive cannot render justified alignment, but this button can set this style in the content.
Insert an ordered list or change an unordered list to an ordered list.  This item has a popup menu that allows you to select the list type.	List	OrderedList	Ordered List	N/A		O	Popup menu was added in EditLive 9.
Insert an unordered list or change an ordered list to an unordered list.  This item has a popup menu that allows you to select the list type.	List	UnorderedList	Unordered List	N/A		T	Popup menu was added in EditLive 9.
Insert an ordered list or change an unordered list to an ordered list.  This item does <i>not</i> have a popup. Only the default ordered list type can be set.	N/A	DefaultOrderedList	Ordered List	N/A		O	Added in EditLive 9.0.0.119
Insert an unordered list or change an ordered list to an unordered list.  This item does <i>not</i> have a popup. Only the default unordered list type can be set.	N/A	DefaultUnorderedList	Unordered List	N/A		T	Added in EditLive 9.0.0.119
Decrease the paragraph or list indent.	N/A	DecreaseIndent	Decrease Indent	N/A		D	
Increase the paragraph or list indent.	N/A	IncreaseIndent	Increase Indent	N/A		N	
The text color selector.	N/A	Color	Color	N/A		C	
The text highlight color selector.	N/A	HighlightColor	Highlight Color	N/A		C	
Superscript text.	Script	Superscript	Superscript	N/A		S	
Subscript text.	Script	Subscript	Subscript	N/A		S	
Strikethrough text.	N/A	Strike	Strikethrough	N/A		S	
Set dir attribute on selected block elements to rtl	textdirection	textrtl	Right to Left Text	N/A		N/A	
Set dir attribute on selected block elements to ltr	textdirection	textltr	Left to Right Text	N/A		N/A	
Remove formatting.	N/A	RemoveFormatting	Remove Formatting	CTRL + SPACE		R	
Format painter.	N/A	FormatPainter	Format Painter	CTRL + SHIFT + C		N/A	
Popup menu of formatting commands, consisting of: <ul style="list-style-type: none"> <li>• Bold</li> <li>• Italic</li> <li>• Underline</li> <li>• Superscript</li> <li>• Subscript</li> <li>• Strikethrough.</li> </ul>	N/A	textFormattingMenu	Text Formatting	N/A		N/A	

Popup menu of Alignment and Indenting commands, consisting of: <ul style="list-style-type: none"> <li>• Align Left</li> <li>• Align Center</li> <li>• Align Right</li> <li>• Decrease Indent</li> <li>• Increase Indent</li> </ul>	N/A	alignIndentMenu	Alignment and Indents	N/A		N/A	
--	-----	-----------------	-----------------------	-----	---	-----	--












# Tool Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Run the spell checker.	Spelling	Spelling...	F7		S
Turn the background spell checker on and off.	BackgroundSpellChecking	Enable/ Disable Check Spelling As You Type	N/A		B
Toggles the Auto Correct functionality on and off.	AutoCorrect	Enable / Disable Auto Correct	N/A	N/A	N/A
Run the thesaurus	Thesaurus	Thesaurus...	N/A	N/A	T
Perform a word count on the document.	WordCount	Word Count...	N/A		W
Get synonyms of a word.	Synonyms	Synonyms	N/A	N/A	N/A
Popup menu of Proofing commands, consisting of: <ul style="list-style-type: none"> <li>• Spelling</li> <li>• Disable Spell Checking as you type</li> <li>• Link Checking</li> <li>• Accessibility Report</li> <li>• Enable Accessibility as you type</li> <li>• Word Count</li> </ul>	CheckerMenu	Proofing Tools	N/A		N/A
Mark selected content as written in chosen language.  Available languages can be defined using the <code>&lt;contentLanguages&gt;</code> element.	contentLanguage	Content Language	N/A		N/A

If using the Popup menu for Proofing Tools, you must ensure that you have included the following plugins in your EditLive! configuration file. For more information please see the [<plugin> Configuration Element](#) article.

- *BrokenHyperlinkReport*
- *accessibility*
- *spelling*


# Table Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Insert a table wizard. Allows user to click and select number of rows and columns. <b>This item can only be used as a toolbar button.</b>	InsTableWizard	Insert Table	N/A		I
Insert a table.	InsTable	Insert Table...	N/A		I
Insert a row in the current table.	InsRow	Insert Row	N/A		N/A
Insert a column in the current table.	InsCol	Insert Column	N/A		N/A
Insert rows or columns in the table.	InsRowCol	Insert Row or Column...	N/A	N/A	R
Insert a cell in a table.	InsCell	Insert Cell	N/A	N/A	E
Delete the current table	DelTable	Delete Table	N/A		T
Delete a row from a table.	DelRow	Delete Row	N/A		D
Delete a column from a table.	DelCol	Delete Column	N/A		C
Delete a cell from a table.	DelCell	Delete Cell	N/A	N/A	L
Split a cell in a table.	Split	Split Cell	N/A		S
Merge cells in a table.	Merge	Merge Cells	N/A		M
Toggle table gridlines on and off.	Gridlines	Show/Hide Gridlines	N/A		G
Automatically resize a column to fit it's content	tableautofit	Fit to Content	N/A		F

# Properties Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Edit the current cell's properties.	PropCell	Cell Properties...	N/A	N/A	R
Edit the selected row's properties.	PropRow	Row Properties...	N/A	N/A	N/A
Edit the selected column's properties.	PropCol	Column Properties...	N/A	N/A	N/A
Edit the current table's properties.	PropTable	Table Properties...	N/A	N/A	T
Edit the properties of a list.	PropList	List Properties...	N/A	N/A	N/A
Edit the properties of an embedded object.	PropObject	Object Properties...	N/A	N/A	N/A
Edit the properties of a horizontal line.	PropHR	Horizontal Line Properties...	N/A	N/A	N/A
Edit the properties of a Section	PropSection	Section Properties...	N/A	N/A	N/A
Edit the properties of the selected Media	PropMedia	Media Properties...	N/A	N/A	M

# Help Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Load EditLive! Help	showHelp	Help...	N/A	?	N/A
Display information about current EditLive! version.	eljAboutELJ	About EditLive!...	N/A		N/A
Popup menu of help and debugging commands, consisting of: <ul style="list-style-type: none"> <li>• Help</li> <li>• About EditLive!</li> <li>• Enable Debug Logging</li> </ul>	helpMenu	Help	N/A	?	N/A

EditLive! help files are, by default, launched from the Tiny website.

## Hosting Help Locally

To customize EditLive! to load Help files from a specific location, perform the following steps:

1. [Contact Tiny support](#) to get a copy of the EditLive! Help source files.
2. Unzip the source files to a desired location in your environment.
3. Add a value parameter to your `<menuItem>` or `<toolbarButton>` elements which specify a **name** attribute of `showHelp`.
  - The value parameter specifies the location of your local Help deployment. Help is internationalized based on the current locale being used by the editor. As such, there are two different types of URLs you can pass to value:
    - a. An absolute URL which will be automatically completed based on the editor's locale. Locale for the editor is either specified explicitly via the [Locale](#) load-time property or automatically detected. Using this method, the location of Help will be constructed by combining the URL in value, the locale and index.html in the following format: `VALUE / LOCALE / index.html`

### Example

The value attribute contains <http://server/docs/help>

The locale is en (which is English)

The editor will load Help from the following location: <http://serverdocs/help/en/index.html>

- a. A relative URL, relative to the [setDownloadDirectory Method](#).

### Example

If you are currently running the editor with an English locale (i.e. en), the following configuration will load Help from <http://www.myserver.com/editlive/enduserhelp/en/index.html>.

```
...
<menuItem name="showHelp" value="http://www.myserver.com/editlive/enduserhelp/" />
...
```

### Example

The following configuration will load Help from the location specified, relative to the [setDownloadDirectory Method](#).

If you are currently running the editor with an English locale (i.e. en), with a DownloadDirectory of <http://www.myserver.com/editlive>, then Help will load from <http://www.myserver.com/editlive/enduserhelp/en/index.html>.

```
...
<toolbarButton name="showHelp" value="enduserhelp/" />
...
```

# Form Commands

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Insert a form.	InsForm	Insert Form	N/A	N/A	F
Insert a text field into a form.	InsTextField	Insert Text Field	N/A	N/A	T
Insert a password field into a form.	InsPasswordField	Insert Password Field	N/A	N/A	P
Insert a hidden field into a form.	InsHiddenField	Insert Hidden Field	N/A	N/A	H
Insert a file browsing field into a form.	InsFileField	Insert File Upload Field	N/A	N/A	I
Insert a button into a form.	InsButtonField	Insert Button Field	N/A	N/A	B
Insert a submit button into a form.	InsSubmitField	Insert Submit Field	N/A	N/A	S
Insert a reset button into a form.	InsResetField	Insert Reset Field	N/A	N/A	R
Insert a checkbox into a form.	InsCheckboxField	Insert Checkbox Field	N/A	N/A	C
Insert a radio-button into a form.	InsRadioField	Insert Radio Field	N/A	N/A	A
Insert a text area into a form.	InsTextAreaField	Insert TextArea Field	N/A	N/A	E
Insert a selection / combobox into a form.	InsSelectField	Insert Select Field	N/A	N/A	L
Insert an image field into a form.	InsImageField	Insert Image Field	N/A	N/A	I
Display the Form Properties Dialog for a form.	PropForm	Form Properties...	N/A	N/A	N/A

# Track Changes Commands

In order for users to utilize the track changes functionality, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Enable tracking changes/Disable tracking changes	enableTrackChanges	Enable Track Changes / Disable Track Changes	CTRL + SHIFT + E		T
Accept Selected Change	acceptChange	Accept Change	N/A		A
Reject Selected Change	rejectChange	Reject Change	N/A		R
Accept All Changes	acceptAllChanges	Accept All Changes	N/A	N/A	E
Reject All Changes	rejectAllChanges	Reject All Changes	N/A	N/A	L
Select Next Change	nextChange	Next Change	N/A		N
Select Previous Change	previousChange	Previous Change	N/A		P
Show Track Changes Dialog	showTrackChangesDialog	Show Track Changes Dialog	N/A	N/A	V
Set Current User	setUsername	Set Username...	N/A	N/A	S











# Image Editor Commands

Enabling Image Editing requires several changes to your EditLive! Configuration File. Please see the [Image Editing](#) article for more information.



In order for users to utilize the Image Editing functionality, one of the following conditions must be met:

- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.








Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Rotate image clockwise	rotateCW	Rotate Clockwise	N/A		N/A
Rotate image counterclockwise	rotateCCW	Rotate Counterclockwise	N/A		N/A
Flip image vertically	flipVertical	Flip Vertically	N/A		N/A
Flip horizontally	flipHorizontal	Flip Horizontally	N/A		N/A
Apply reflection effect	reflect	Reflection Effect	N/A		N/A
Apply drop-shadow effect	dropShadow	Drop Shadow Effect	N/A		N/A
Apply rounded-corners effect	roundedCorners	Round Corners Effect	N/A		N/A
Crop Image	crop	Crop	N/A		N/A

# Accessibility Commands

Enabling accessibility checking requires several changes to your EditLive! Configuration File. For more information, please see the [Table Accessibility](#) and [Accessibility As You Type](#) articles.

In order for users to utilize the Accessibility functionality, one of the following conditions must be met:

- Users are still operating the editor in their 30 day trial period, or
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.


Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Accessibility-As-You-Type renders icons on in-accessible content items	AccessibilityAsYouType	Enable / Disable Accessibility As You Type	N/A		N/A
Perform an accessibility compliance check on the document, based on the W3C standards.	Accessibility	Accessibility Report...	CTRL + F8		A
Change all widths and heights in a table to percentages.	percentageTableSizing	Use Relative Sizing for Table	N/A		N/A
Change all widths and heights in a table to pixels.	pixelTableSizing	Use Pixel Sizing for Table	N/A		N/A
Clear table cell headers	clearCellHeaders	Clear Cell Headers	N/A		N/A
Apply table cell headers	applyCellHeaders	Set As Header / Select Data Cells / Apply Header	N/A		N/A
Display table header-to-data cell mappings	tableHeaderMappings	Show / Hide Table Header Mappings	N/A		N/A

# Broken Hyperlink Report

Enabling the Broken Hyperlink Report also requires a change to your EditLive! Configuration File. For more information please see the [<plugin> Configuration Element](#) article.

In order for users to utilize the Broken Hyperlink Report functionality, one of the following conditions must be met:

- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Check hyperlinks throughout the document to ensure they're valid	BrokenHyperlinkReport	Broken Hyperlink Report	N/A		B


# Commenting Commands

Enabling Commenting requires a change to your EditLive! Configuration File. For more information please see the [<plugin> Configuration Element](#) article.




In order for users to utilize the Commenting functionality, one of the following conditions must be met:

- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.


Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Open commenting popup to add or read comments on a selected area of text	AddComment	Add Comment...	N/A		N/A
Remove all comments in the document	RemoveAllComments	Remove All Comments	N/A	N/A	N/A

# Equation Editor Commands

Enabling the Equation Editor requires sever changes to your EditLive! Configuration File. For more information, see the [Equation Editor Add-On](#) article.

 In order for users to utilize the equation editing functionality, one of the following conditions must be met:

- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
Insert a MathML mathematical equation.	InsertEquation	Insert Equation	N/A		E
Edit a MathML mathematical equation	EditEquation...	Edit Equation...	N/A	N/A	N/A

# Menu and Toolbar Item Groups

This collection of interface commands can be used only within menus and toolbars in EditLive!. These interface items are added as a group of buttons or menu items. The activation of buttons and menu items in the groups is mutually exclusive; for example, the left alignment item cannot be activated at the same time as the center or right alignment buttons. The items, their corresponding function, tool tip and menu text, mnemonic, shortcuts, images, and their associated value for the relevant name attribute of the group are listed below.

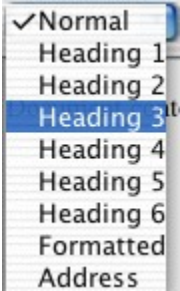
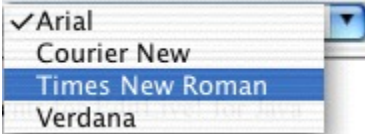
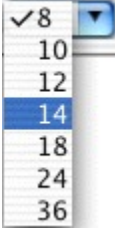
## Submenu Items

The following predefined submenus can be used in any menu of the EditLive! interface.

Function	XML Name Attribute	Submenu Item Name	Mnemonic
List the available fonts.	FontFace	Font	F
List the available font sizes.	FontSize	Size	S
List the available styles.	Style	Style	T
List the available text foreground colors.	Color	Color	C
List the available highlighter colors.	HighlightColor	Highlight Color	C
List the elements available for selection according to the	Select	Select	S

## Toolbar Combo Items

The following predefined combo box items are available for use on the EditLive! toolbars. The values displayed in these combo box items are specified by using child <comboBoxItem> elements.

Function	XML Name Attribute	Example Image
List of styles available for use in this document.	Style	
List of fonts available for use in this document.	Face	
List of font sizes available for use in this document.	Size	

# Creating Custom Menu and Toolbar Items

Tiny EditLive! allows developers to create custom menu and toolbar items. Developers can specify the text, image, and functionality of these menu and toolbar items.

## Custom Toolbar Items

There are two types of custom items that can be added to the EditLive! toolbar:

- Custom Toolbar Button Items
- Custom Toolbar Combo Items

### Custom Toolbar Button Items

Custom toolbar buttons are specified using the `<customToolbarButton>` configuration file element. Through this configuration file element developers can specify the image, tooltip text, and operation associated with the custom toolbar button.

#### Example

The following configuration file example demonstrates how to define a custom toolbar button for use within EditLive! on the Command Toolbar. The button defined in this example will insert the HTML `<p>HTML to insert</p>` at the cursor.

HTML to be inserted at the cursor must be [URL encoded](#). For this example the value to be inserted has already been URL encoded.

```
<editLive>
...
<toolbars>
  <toolbar name="command">
    <customToolbarButton
      name="customButton1"
      text="Custom Button"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="insertHTMLatCursor" value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </toolbar>
  </toolbars>
...
</editLive>
```

For additional information and examples of custom toolbar buttons, please see the [<customToolbarButton>](#) configuration file article.

### Custom Toolbar Combobox Items

Custom toolbar combo boxes are specified using the `<customToolbarComboBox>` configuration file element. The custom toolbar combo box item itself can be configured to perform custom operations, and/or its nested combo box items can be configured to perform custom operations. Combo box items to be contained within a custom combo box are specified using the `<customComboBoxItem>` configuration file element.

#### Example

The following example specified a custom toolbar combo box item. This combo box itself doesn't perform a custom operation, but its child custom combo box item performs an insert HTML at cursor custom operation, `<p>HTML to insert</p>`.

HTML to be inserted at the cursor must be [URL encoded](#). For this example the value to be inserted has already been URL encoded.

```
<editLive>
...
<toolbars>
  <toolbar name="command">
    <customToolbarComboBox name="customCombo">
      <customComboBoxItem
        name="customComboItem1"
        text="Custom Combo Item"
        action="insertHTMLatCursor"
        value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
      </customToolbarComboBox>
    </toolbar>
  </toolbars>
...
</editLive>
```

For additional information and examples of custom toolbar buttons, please see the [<customToolbarComboBox>](#) and [<customComboBoxItem>](#) configuration file articles.

## Custom Menu Items

Custom menu items in EditLive! can only be of one format and of a single layered depth (i.e. they cannot include submenus). Custom menu items can be added to any of the menus in EditLive! via the `<customMenuItem>` configuration file.

### Example

The following configuration file example demonstrates how to define a custom menu item for use within EditLive!'s Insert menu. The button defined in this example will insert the HTML `<p>HTML to insert</p>` at the cursor.

HTML to be inserted at the cursor must be [URL encoded](#). For this example the value to be inserted has already been URL encoded.

```
<editLive>
...
<menuBar>
...
<menu name="ephox_insertmenu">
  <customMenuItem
    name="customItem1"
    text="Custom Item"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="insertHTMLAtCursor" value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
  </menu>
...
</menuBar>
...
</editLive>
```

Custom menu items can also be embedded into the shortcut menu. For example:

```
<editLive>
...
<shortcutMenu>
  <shrtMenu>
    ...
    <customMenuItem
      name="customItem1"
      text="Custom Item"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="insertHTMLAtCursor" value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    ...
  </shrtMenu>
</shortcutMenu>
...
</editLive>
```

## Custom Operations for Toolbar and Menu Items

Custom menu and toolbar items can be used to perform the following operations:

- Insert HTML at the cursor
- Insert a hyperlink at the cursor
- Raise a JavaScript event
- Calling a JavaScript function, passing the current tag's attributes
- Cause EditLive! to HTTP Post its content to a specific URL.

### Inserting HTML and Hyperlinks at the Cursor

When configuring custom toolbar buttons, combo boxes, and menu items within EditLive! to insert HTML at the cursor, the HTML needs to be specified within the EditLive! configuration file. The HTML to insert in the XML configuration file the HTML also needs to be already URL encoded.

The insert hyperlink at cursor custom functionality in EditLive! requires that the user select text before using the relevant custom item. Upon using the relevant custom item, the selected text will become a hyperlink linking to the address specified in the custom item configuration.

The HTML or hyperlink to insert at the location of the cursor is specified via the **value** attribute of the related `<customComboBoxItem>`, `<customToolbarButton>` or `<customMenuItem>` element in the EditLive! configuration file.

### Example

The following example demonstrates how to define a custom menu item which uses the value `insertHTMLAtCursor` for the **action** attribute to insert HTML into an instance of EditLive!. The HTML to insert is the string `<p>HTML to insert</p>`, which is already URL encoded and stored in the **value** attribute.



```

<editLive>
...
<menuBar>
...
  <menu name="ephox_insertmenu">
    <customMenuItem
      name="customItem1"
      text="Custom Item"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="insertHTMLAtCursor"
      value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </menu>
  ...
</menuBar>
...
</editLive>

```

### Example

The following example demonstrates how to define a custom menu item which uses the value *insertHyperlinkAtCursor* for the **action** attribute to insert a hyperlink into an instance of EditLive!. The hyperlink to insert is the URL *http://www.ephox.com*, which is stored in the **value** attribute.

```

<editLive>
...
<menuBar>
...
  <menu name="ephox_insertmenu">
    <customMenuItem
      name="customItem2"
      text="Ephox"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="insertHyperlinkAtCursor"
      value="http://www.ephox.com" />
    </menu>
  ...
</menuBar>
...
</editLive>

```

## Raising a JavaScript Event

Custom toolbar and menu items in EditLive! can be configured to raise JavaScript events. JavaScript events raised through EditLive! are required to be defined either in the page in which EditLive! is embedded or in a file which is included in the page EditLive! is embedded in.

To raise a JavaScript event through a `<customComboBoxItem>`, `<customToolbarButton>` or `<customMenuItem>` element, the value *raiseEvent* is used for the **action** attribute. The **value** attribute of the element should specify the name of the JavaScript function which is to be called.

### Example

The following example demonstrates how to define a custom menu item which uses the *raiseEvent* action for use within EditLive!. The menu item defined in this example will call the JavaScript function called *eventRaised*.

```

<editLive>
...
<menuBar>
...
  <menu name="ephox_insertmenu">
    <customMenuItem
      name="customItem1"
      text="Raise Event"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="raiseEvent"
      value="eventRaised" />
    </menu>
  ...
</menuBar>
...
</editLive>

```

## Calling a JavaScript Function, Passing the Current Tag's Attributes

Custom toolbar and menu items in EditLive! can be configured to pass all the attributes of the current tag to a specified JavaScript function. Developers can use this functionality with their own JavaScript functions to manipulate the attributes of specific tags at runtime. This allows developers to create custom dialogs which pertain to specific tags. This is useful in several ways: providing new dialogs for HTML elements (e.g. `<span>`, which is not available for direct interaction within EditLive!); accessing the properties of custom or XML tags such as `<custom>`; or replacing/complementing an existing EditLive! dialog, such as the Image Properties dialog which corresponds with the `<img>` tag.

To pass the attributes of the current tag to a JavaScript function through a `<customComboBoxItem>`, `<customToolbarButton>` or `<customMenuItem>` element, the value `customPropertiesDialog` is used for the **action** attribute. The **value** attribute of the element should specify the name of the JavaScript function to be called.

The JavaScript function called will have a string parameter passed to it. This string will be a collection of name-value pairs for each attribute found in the tag. The string will also contain an `ephoxID` name-value pair. This `ephoxID` value is used by EditLive! to keep track of all tags currently stored in the editor.

Tiny strongly advises developers **not** to change the value of the `ephoxID` name-value pair.

### Example

The following example demonstrates how to define a custom menu item which uses the `customPropertiesDialog` action for use within EditLive!. The menu item defined in this example will call the JavaScript function called `DisplayAttributes`. This menu item is only available when a `<h1>` tag is selected.

This operation will only work for one specified tag type. To specify which tag type the menu or toolbar item will appear for, use the **enableintag** attribute.

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="ephox_insertmenu">
      <customMenuItem
        name="showAttributes"
        text="H1 Properties"
        action="customPropertiesDialog"
        value="customPropDemo"
        enableintag="h1"
      />
    </menu>
  </menuBar>
  ...
</editLive>
```

For an instance of EditLive! using the `<customMenuItem>` created above, the following function would create a JavaScript dialog displaying each name-value pair of attributes for the `<h1>` tag.

```
function customPropDemo(properties)
{
  alert("VALUE-NAME pairs: " + properties);
}
```

The **setProperties Method** of EditLive! is used to set new attribute values for a tag. This property takes a string containing all the new name-value pairs for the tag to be changed. Which tag to change is specified through the `ephoxID` name-value pair.

## Generating a HTTP Post for EditLive!'s Content

Custom toolbar and menu items in EditLive! can be configured to cause EditLive! to POST its content to a specific URL.

To post the content of EditLive! to a server-side script event through a `<customComboBoxItem>`, `<customToolbarButton>` or `<customMenuItem>` element, the value `PostDocument` is used for the **action** attribute.

The **value** attribute of the element can be used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the **value** attribute:

### Post Field

The name of the field in the HTTP POST that EditLive! uses to POST its content.

This parameter is required.

### Post Acceptor URL

The URL for the POST acceptor that EditLive! is to POST to.

The parameter is required.

### Response Processing

The operation that EditLive! is to perform with the HTTP response from the POST acceptor script.

The parameter can have the following values:

- `saveToDisk` - Present the user with a save file dialog, with which they can save the response to the local machine.
- `callback` - Pass the entire content of the HTTP response to a specified JavaScript callback function for processing.

This parameter is required.

#### JavaScript Callback Function

The name of the JavaScript callback function to use for processing the response.

This parameter should only be used if the response processing is set to `callback`.

The parameters specified through the **value** attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing and JavaScript Callback Function (if needed).

#### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called `POST_field`, sending the contents via HTTP Post to `http://someserver/postacceptor.jsp`, then call back the JavaScript function called `JSFunction`.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

#### Example

The following example demonstrates how to define a custom menu item which uses the `PostDocument` action for use within EditLive!. The menu item defined in this example will POST the content in the field `editlive_field` to the script at `http://someserver/post/POSTacceptor.aspx` upon completion of the POST the content of the HTTP response will be passed to the JavaScript callback function `JSFunction`.

```
<editLive>
...
<menuBar>
...
<menu>
  <customMenuItem
    name="customItem1"
    text="POST Content"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="PostDocument" value="editlive_field##ephox##http://someserver/post/POSTacceptor.aspx
##ephox##callback##ephox##JSFunction" />
  </menu>
...
</menuBar>
...
</editLive>
```

## See Also

- [<customMenuItem>](#)
- [<customToolBarButton>](#)
- [<customToolBarComboBox>](#)
- [<customComboBoxItem>](#)
- [Encoding Content for Use with EditLive!](#)

# Customizing the Color Picker

EditLive! uses the same color chooser dialog throughout the editor. This color chooser dialog can be seen when using the following functionality in the editor:

- Selecting the foreground/background color for text
- Selecting the background color for table cells, rows, columns
- Selecting the color for *HR* HTML elements

This color chooser dialog can be customized by editing the [Configuration File](#) used by the editor.

## Color Picker Configuration Elements

The color picker can be configured using the `<colorPalette>` and `<color>` elements respectively. Using these configuration elements, you can specify which colors appear in the color picker as well as limit users from specifying their own custom colors.

### Example

The following XML in an EditLive! [Configuration File](#) would display the colors red, blue, and yellow in the color chooser used by the editor.

Colors must be specified using their hexadecimal representation (e.g. "#1234AB").

```
...  
  
<wysiwygEditor>  
  <colorPalette>  
    <color name="#FF0000" />  
    <color name="#0000FF" />  
    <color name="#FFFF00" />  
  </colorPalette>  
</wysiwygEditor>  
  
...
```

### See Also

- [Manually Editing Configuration Files](#)
- [EditLive! Configuration File Elements](#)
- [<colorPalette> Configuration Element](#)
- [<color> Configuration Element](#)

# Cascading Stylesheet Support

- [Using CSS in EditLive!](#)
  - [Using CSS in the Applet](#)
  - [Using CSS in the Swing SDK](#)
- [Using CSS Extensions to Render Custom Tags](#)
- [Restricting what CSS classes appear in EditLive! styles drop down](#)

# Using CSS in EditLive!

# Using CSS in the Applet

EditLive! provides support for the use of Cascading Style Sheets (CSS) to enforce formatting standards easily by separating content from formatting. Styles can be specified three ways: via an external, linked style sheet; through an embedded style sheet; or through inline style information (e.g. in a <SPAN> tag). This article assumes the reader is familiar with the concept of using Cascading Style Sheets. If you would like to learn more about CSS before reading this document, please visit the [W3C's Introduction to CSS](#).

## Style Types

EditLive! provides several methods of adding style information to a document. Furthermore, EditLive! recognizes style information and populates the style drop-down list box accordingly. EditLive! can apply styles both as inline styles and block styles. Block styles are applied to an entire XHTML element such as a <P> tag, whereas inline styles are applied to a section of text within a XHTML element.

### Specifying Block Styles

The way in which styles are specified within a style sheet affects the way they can be applied within EditLive!. Style classes which are directly associated with a block tag can only be applied to a block tag. These styles are designated by a ¶ symbol on the styles drop-down in EditLive!. For example, a block style which defines that paragraph text should be blue can be defined as follows:

```
p.blue{color: blue}
```

### Specifying Inline Styles

Inline styles in EditLive! are applied using the <SPAN> XHTML element. Thus, a style which can only be used inline should be defined as a class to be used with the <SPAN> tag. These styles are designated by a ¶ symbol on the styles drop-down in EditLive!. An inline style which would specify text that is to have a red color would be as follows:

```
span.redtext{color: red}
```

### Specifying a Block and Inline Style

Finally, it is possible to define a style class which may be used as both an inline and block style. These styles appear twice on the EditLive! styles drop-down - once marked with the ¶ symbol, and again with an ¶ symbol. A style class which could be applied on both block and inline tags to change the text color to green is as follows:

```
.green{color: green}
```

## User Friendly Style Names

EditLive! will attempt to display a more user friendly name of the defined CSS classes in the styles drop down. If the style name contains hyphens, underscores or uses camel-case naming convention the name will be displayed using spaces instead of the hyphen or underscore or split using spaces for camel case.

### Example

- "p.strong-emphasis" will become "Paragraph (Strong Emphasis)"
- "p.red\_blackground" will become "Paragraph (Red Background)"
- "p.blueText" will become "Paragraph (Blue Text)"

## Setting Styles in EditLive!

Style information for EditLive! can be set using the following methods:

- **Using the EditLive! Configuration File to Set Styles**  
A configuration file may include style information in the <head> element in a couple of ways:
  - An external style sheet may be specified through the use of the <link> element, or
  - An embedded style sheet can be specified through the <style> element.

With the exception of inline styles, style information specified through the use of a configuration file takes precedence over style information specified in any other way.

- **Importing Styles From Microsoft Word**  
Microsoft Word Styles can be imported from a Microsoft Word document using simple copy and paste. This action is permitted using the <wordImport> element.
- **Setting Styles Via the EditLive! Document Load Time Property**  
Styles may be specified as part of the <head> tag of a HTML document when using the [setDocument Method](#) to set the content of EditLive!.

- **Setting Styles via the EditLive! SetStyles Load Time Property**

The [setStyles Method](#) can be used to explicitly set style information. It should be noted that EditLive! CSS support complies with the W3C CSS precedence rules. Thus, inline styles take precedence over an embedded style sheet. Furthermore, the styles listed in an embedded style sheet take precedence over those from an external style sheet. Finally, when multiple external style sheets are used, style sheets listed last will have precedence if there are any conflicts between style sheets.

If [embedded](#) styles are specified using the a configuration file, the embedded styles specified will be the only embedded styles to exist in the XHTML.

**Example**

If the configuration file specifies

```
H1{font-size: 10;}
```

and the [setStyles Method](#) for an instance of EditLive! specifies

```
H1{font-size: 20} H2{font-size: 15}
```

, then the resulting embedded styles for the XHTML will still only be

```
H1{font-size: 10}
```

[Inline](#) styles will still exist for any single tag specified.

**Example**

If the XHTML specified with an instance of EditLive! contains `<span style="color: yellow">Yellow Text</span>` then this style will still exist for the specified tag.

## Using the EditLive! Configuration File to Set Styles

### Linking to an External Style Sheet

EditLive! can be configured to use external style sheets by specifying a value with the `<link>` element in the configuration file.

**Example**

Linking to the stylesheet `mystyles.css`, located in the same directory as the HTML page implementing this configuration file, will specify that specific style sheet to be used.

```
<editLive>
  <document>
    <html>
      <head>
        <link href="mystyles.css" rel="stylesheet" type="text/css"/>
      </head>
    </html>
  </document>
  ...
</editLive>
```

When EditLive! links to multiple style sheets, the last style sheet added will have priority if there is a conflict with an earlier style sheet.

**Example**

`stylesheet1.css` defined H1 as:

```
H1{font-family: Arial, Helvetica, sans-serif; font-size: 24pt;}
```

`stylesheet2.css`, linked after `stylesheet1.css`, defined H1 as:

```
H1{font-family: Arial, Helvetica, sans-serif; font-size: 48pt; }
```

The value of H1 in `stylesheet1.css` would be overridden by the H1 value given in `stylesheet2.css` (i.e. H1 would be size 48pt, not 24pt). Hence, the order that style sheets are added is important and will effect how the HTML is formatted.



## Defining an Embedded Style Sheet

An embedded style sheet can be specified through the `<style>` element of the EditLive! configuration file. Styles listed in a style sheet embedded via the configuration file take precedence over styles otherwise defined.

### Example

The following would configure EditLive! to use the provided embedded style sheet. The embedded style sheet used for this example would implement a mixture of inline and block styles.

```
<editLive>
  <document>
    <html>
      <head>
        <style>
          <!--
            p.blue{color: blue;}
            span.red{color: red;}
            .green{color: green;}
          -->
        </style>
      </head>
    </html>
  </document>
</editLive>
```

## Importing Styles from Microsoft Word

Styles may be easily imported from Microsoft Word. This action is performed by copying text from Microsoft Word and selecting paste in EditLive!.

Developers can customize the manner in which EditLive! copies styles from Microsoft Word using the `<wordImport>` configuration file element. EditLive! provides three different methods for copying styles from Microsoft Word:

- **Clean** - Copying content from Microsoft Word, adding no style information,
- **Merge Embedded** - Style information is stored in the `<head>` of the document with other embedded styles, and
- **Merge Inline** - Style information is explicitly specified inline for text copied.

In cases where EditLive! uses the configuration file to specify embedded styles, users should copy text and styles from Microsoft Word using Merge Inline. The reason for this is that embedded styles specified in a configuration file override all other embedded styles specified for the XHTML, including styles copied from Microsoft Word.

## Setting Styles via the Document Load Time Property

When instantiating an instance of EditLive!, the `setDocument Method` can be used to populate the editor with a complete HTML document. An embedded style sheet can be specified in the head of this document.

### Example

To create an embedded style sheet specifying H1 to be of font size 20pt, the following methods of EditLive! would be called:

```
<script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELJApplet1", 700, 400);
  editlivejava1.setLocalDeployment(false);
  editlivejava1.setXMLURL("eljconfig.xml");
  editlivejava1.setDocument(encodeURIComponent("<html><head><style><!--H1{FONT-SIZE: 20pt;}--></style><
/head><body><h1>Testing Header Size</h1></body></html>"));
  editlivejava1.show();
</script>
```

Any embedded style sheets defined in the EditLive! configuration file take precedence over those defined with the Document property.

## Setting Styles via the Styles Load Time Property

Styles can be directly provided to EditLive! when it is instantiated. This is achieved by setting the `setStyles Method` of EditLive!. Styles defined in this manner act as an embedded style sheet within EditLive!.

### Example

If the `setStyles Method` or attribute was set to be:

```
editlivejava1.setStyles(encodeURI("H1 { FONT-SIZE: 36pt; }"));
```

then the information would appear in EditLive! for Java's Code View as:

```
<HEAD>
...
<STYLE>
  <!--
    H1 { FONT-SIZE: 36pt; }
  -->
</STYLE>
...
</HEAD>
```

Any embedded style sheets defined in the EditLive! configuration file take precedence over those defined with the Styles property.

## Populating the Styles Drop-Down List Box

The styles drop-down box appearing on the EditLive! will render based on the following conditions:

- The EditLive! [Configuration File](#) needs to specify a `<toolbarComboBox>` element with a **name** attribute containing the string *Style*. Each `<comboBoxItem>` child element in `<toolbarComboBox>` will define the tag types the user can insert.
- The actual CSS loaded into EditLive! uses the methods outlined in [Setting Styles in EditLive!](#).

### Example

A stylesheet loaded into EditLive! defines the following styles:

```
h1{color: yellow}
p.blue{color: blue}
span.red{color: red}
.green{color: green}
```

The EditLive! Configuration File used features the following `<toolbarComboBox>` element:

```
<toolbarComboBox name="Style">
  <comboBoxItem name="P" text="Normal"/>
  <comboBoxItem name="H1" text="Heading 1"/>
</toolbarComboBox>
```

The following would appear in the drop-down styles list box:

Paragraph Styles	
Normal	¶
Heading 1	¶
Normal (blue)	¶
.green	¶
Inline Styles	
.red	Ⓐ
.green	Ⓐ

.green class appears twice as it can be applied as both an inline and a block style.

<comboBoxItem> Configuration File elements can use the text attribute to create an alias for the Style type. In the example above, you can see that H1 tags appear as Heading 1 instead of H1. For more information on creating aliases for styles, see the <comboBoxItem> article.

## Styles Drop-Down Hierarchy

The styles drop down displays available styles according to where the cursor has been inserted. It will display styles in the following order:

- **Paragraph Styles** (e.g. P tag, Heading tags)
- **Inline Styles** (e.g. Span tags )
- **Element Styles** (styles available to the current element selected)
- **Parent Element Styles** (styles for each parent element of the currently selected element).

### Example

The following HTML is loaded into EditLive!:

```
<h1>Table Example</h1>
<table>
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
```

If the user selected the text *Cell 1*, the style drop-down would display the following available styles:

- Paragraph Styles
- Inline Styles
- Cell Styles
- Row Styles
- Table Styles

Styles will appear under their defined type (e.g. paragraph, inline, element) in the following hierarchy:

- Default element styles (e.g. h1)
- All element specific CSS class (e.g. h1.h1Style)
- All generic CSS classes (e.g. .genericStyle)

The order by which default element styles, element specific styles, and generic styles are arranged will depend on the order in which these styles appear when passed to EditLive!.

### Example

A stylesheet loaded into EditLive! defines the following styles:

```
p.blue{color: blue}
p.yellow(color: yellow}
```

```
.green{color: green}
.brown{color: brown}
```

The Styles drop-down will first render the default element styles (i.e. *p.blue*, then *p.yellow*). The generic elements will then be rendered in the order *.green* then *.brown*.

**Example**

A stylesheet loaded into EditLive! defines the following styles:

```
h1{color: yellow}
p.blue{color: blue}
span.red{color: red}
.green{color: green}
```

The EditLive! [Configuration File](#) features the following <toolbarComboBox> element:

```
<toolbarComboBox name="Style">
  <comboBoxItem name="P" text="Normal"/>
  <comboBoxItem name="H1" text="Heading 1"/>
</toolbarComboBox>
```

The following would appear in the drop-down styles list box:

Paragraph Styles	
Normal	¶
Heading 1	¶
Normal (blue)	¶
.green	¶
Inline Styles	
.red	a
.green	a

You can see that the default styles for the <P> and <H1> tags appear first. Then CSS class specific to <P> (*.blue*) appears next. Finally, the generic CSS class *.green* appears last.

**Getting Styles from EditLive!**

Style information for EditLive! can be extracted using the following methods:

- Using the [getStyles Method](#).
- Accessing the `_styles` HTTP Post Attribute
- Using the [getDocument Method](#).  
This method returns the entire XHTML Document, including the embedded styles in the <head> tag.

Accessing the `_styles` HTTP Post Attribute

As documented in the [Retrieving Content From EditLive!](#) article, both the HTML content and the CSS styles contained in an instance of EditLive! can be sent through a HTTP Post. When a form containing an instance of EditLive! is submitted, a hidden field containing all style information for the document is also created and posted with the form data. This hidden field will be called `ELAppletName_styles`, where `ELAppletName` is the name specified by the developer in the EditLiveJava constructor.

### Example

The following example highlights naming an instance of EditLive! *ELAppletName*.

```
<script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELAppletName", 700, 400);
  editlivejava1.setLocalDeployment(false);
  editlivejava1.setConfigurationFile("eljconfig.xml");
  editlivejava1.show();
</script>
```

The page listed as the **action** to receive the HTTP Post could then contain controls to assign the styles information to a variable, which could then be saved to a back-end database.

### Example

The page sending the HTTP Post has generated a hidden field called *ELAppletName\_styles*, containing the styles information.

### VB Scripting

```
<%
  Dim stylesInformation = Request( "ELAppletName_styles" )
%>
```

### JSP Scripting

```
<%
  String stylesInformation = request.getParameter("ELAppletName_styles");
%>
```

### See Also

- [Retrieving Content From EditLive!](#)
- [<head> Configuration File Element](#)
- [<link> Configuration File Element](#)
- [<wordImport> Configuration File Element](#)

# Using CSS in the Swing SDK

EditLive! for Java Swing provides support for the use of Cascading Style Sheets (CSS) to enforce formatting standards easily by separating content from formatting. Styles can be specified three ways: via an external, linked style sheet; through an embedded style sheet; or through inline style information (e.g. in a <SPAN> tag). This article assumes the reader is familiar with the concept of using Cascading Style Sheets. If you would like to learn more about CSS before reading this document, please visit the [W3C's Introduction to CSS](#).

## Style Types

EditLive! for Java Swing provides several methods of adding style information to a document. Furthermore, EditLive! for Java Swing recognizes style information and populates the style drop-down list box accordingly. EditLive! for Java Swing can apply styles both as inline styles and block styles. Block styles are applied to an entire XHTML element such as a <P> tag, whereas inline styles are applied to a section of text within a XHTML element.

### Specifying Block Styles

The way in which styles are specified within a style sheet affects the way they can be applied within EditLive! for Java Swing. Style classes which are directly associated with a block tag can only be applied to a block tag. These styles are designated by a ¶ symbol on the styles drop-down in EditLive! for Java Swing. For example, a block style which defines that paragraph text should be blue can be defined as follows:

```
p.blue{color: blue}
```

### Specifying Inline Styles

Inline styles in EditLive! for Java Swing are applied using the <SPAN> XHTML element. Thus, a style which can only be used inline should be defined as a class to be used with the <SPAN> tag. These styles are designated by a ¶ symbol on the styles drop-down in EditLive! for Java Swing. An inline style which would specify text that is to have a red color would be as follows:

```
span.redtext{color: red}
```

### Specifying a Block and Inline Style

Finally, it is possible to define a style class which may be used as both an inline and block style. These styles appear twice on the EditLive! for Java Swing styles drop-down - once marked with the ¶ symbol, and again with an ¶ symbol. A style class which could be applied on both block and inline tags to change the text color to green is as follows:

```
.green{color: green}
```

## Making Style names more readable

EditLive! will attempt to display a more human readable name of the defined CSS styles for display in the styles drop down. If the style name contains hyphens, underscores or uses camel-case naming convention the name will be displayed using spaces instead of the hyphen or underscore or split using spaces for camel case.

### Example

- "p.strong-emphasis" will become "Paragraph (Strong emphasis)"
- "p.red\_blackground" will become "Paragraph (Red background)"
- "p.blueText" will become "Paragraph (Blue text)"

## Setting Styles in EditLive! for Java Swing

Style information for EditLive! for Java Swing can be set using the following methods:

- **Using the EditLive! for Java Swing Configuration File to Set Styles**  
A configuration file may include style information in the <head> element in a couple of ways:
  - An external style sheet may be specified through the use of the <link> element, or
  - An embedded style sheet can be specified through the <style> element.

With the exception of inline styles, style information specified through the use of a configuration file takes precedence over style information specified in any other way.

- **Importing Styles From Microsoft Word**  
Microsoft Word Styles can be imported from a Microsoft Word document using simple copy and paste. This action is permitted using the <wordImport> element.
- **Setting Styles via the setDocument method**  
Styles may be specified as part of the <head> tag of a HTML document when using the [setDocument\(\)](#) method of the ELJBean class to set the content of EditLive! for Java Swing. Styles can also be set when passing the desired HTML document to ELJBean constructor as a String.

- **Using the setStyles method**

The `setStyles()` method of the `ELJBean` class can be used to specify CSS for the content. It should be noted that `EditLive!` for Java Swing CSS support complies with the W3C CSS precedence rules. Thus, inline styles take precedence over an embedded style sheet. Furthermore, the styles listed in an embedded style sheet take precedence over those from an external style sheet. Finally, when multiple external style sheets are used, style sheets listed last will have precedence if there are any conflicts between style sheets.

If embedded styles are specified using the a configuration file, the embedded styles specified will be the only embedded styles to exist in the XHTML.

**Example**

If the configuration file specifies

```
H1{font-size: 10;}
```

and the `setStyles Method` for an instance of `EditLive!` specifies

```
H1{font-size: 20} H2{font-size: 15}
```

, then the resulting embedded styles for the XHTML will still only be

```
H1{font-size: 10}
```

Inline styles will still exist for any single tag specified.

**Example**

If the XHTML specified with an instance of `EditLive!` for Java Swing contains `<span style="color: yellow">Yellow Text</span>` then this style will still exist for the specified tag.

## Using the EditLive! Configuration File to Set Styles

### Linking to an External Style Sheet

`EditLive!` can be configured to use external style sheets by specifying a value with the `<link>` element in the configuration file.

**Example**

Linking to the stylesheet `mystyles.css`, located in the same directory as the HTML page implementing this configuration file, will specify that specific style sheet to be used.

```
<editLive>
  <document>
    <html>
      <head>
        <link href="mystyles.css" rel="stylesheet" type="text/css"/>
      </head>
    </html>
  </document>
  ...
</editLive>
```

When `EditLive!` for Java Swing links to multiple style sheets, the last style sheet added will have priority if there is a conflict with an earlier style sheet.

**Example**

`stylesheet1.css` defined H1 as:

```
H1{font-family: Arial, Helvetica, sans-serif; font-size: 24pt;}
```

`stylesheet2.css`, linked after `stylesheet1.css`, defined H1 as:

```
H1{font-family: Arial, Helvetica, sans-serif; font-size: 48pt; }
```

The value of H1 in `stylesheet1.css` would be overridden by the H1 value given in `stylesheet2.css` (i.e. H1 would be size 48pt, not 24pt). Hence, the order that style sheets are added is important and will effect how the HTML is formatted.

## Defining an Embedded Style Sheet

An embedded style sheet can be specified through the `<style>` element of the `editLive!` for Java Swing configuration file. Styles listed in a style sheet embedded via the configuration file take precedence over styles otherwise defined.

### Example

The following would configure `editLive!` for Java Swing to use the provided embedded style sheet. The embedded style sheet used for this example would implement a mixture of inline and block styles.

```
<editLive>
  <document>
    <html>
      <head>
        <style>
          <!--
            p.blue{color: blue;}
            span.red{color: red;}
            .green{color: green;}
          -->
        </style>
      </head>
    </html>
  </document>
</editLive>
```

## Importing Styles from Microsoft Word

Styles may be easily imported from Microsoft Word. This action is performed by copying text from Microsoft Word and selecting paste in `editLive!` for Java Swing.

Developers can customize the manner in which `editLive!` for Java Swing copies styles from Microsoft Word using the `<wordImport>` configuration file element. `editLive!` for Java Swing provides three different methods for copying styles from Microsoft Word:

- **Clean** - Copying content from Microsoft Word, adding no style information,
- **Merge Embedded** - Style information is stored in the `<head>` of the document with other embedded styles, and
- **Merge Inline** - Style information is explicitly specified inline for text copied.

In cases where `editLive!` for Java Swing uses the configuration file to specify embedded styles, users should copy text and styles from Microsoft Word using **Merge Inline**. The reason for this is that embedded styles specified in a configuration file override all other embedded styles specified for the XHTML, including styles copied from Microsoft Word.

## Setting Styles via the setDocument Method

For an instance of the `ELJBean` class, the `setDocument()` method can be used to populate the editor with a complete HTML document. An embedded style sheet can be specified in the head of this document.

### Example

To create an embedded stylesheet specifying H1 to be of font size 20pt, the following method of `ELJBean` would be called:

```
ELJBean myEditor = new ELJBean();
myEditor.setDocument("<html><head><style><!--H1{FONT-SIZE:20pt;}--></style>" +
  "</head><body><h1>Testing Header Size</h1></body></html>");
```

Any embedded style sheets defined in the `editLive!` for Java Swing configuration file takes precedence over those defined with the `setDocument()` method.

## Using the setStyles Method

Styles can be directly provided to an `ELJBean` instance by using the `setStyles()` method. Styles defined in this manner act as an embedded style sheet within `editLive!` for Java Swing.

### Example

If the `setStyles` method was set to be:

```
ELJBean myEditor = new ELJBean();

/*
  ... setDocument method called to set body
*/

myEditor.setStyles("H1{FONT-SIZE:36pt;}");
```



Then the information would appear in the EditLive! for Java Swing Code View as:

```
<HEAD>
...
<STYLE>
  <!--
  H1 { FONT-SIZE: 36pt; }
  -->
</STYLE>
...
</HEAD>
```

Any embedded style sheets defined in the EditLive! for Java Swing configuration file take precedence over those defined with the setDocument() method.

## Populating the Styles Drop-Down List Box

The styles drop-down box appearing on the EditLive! will render based on the following conditions:

- The EditLive! for Java Swing [Configuration File](#) needs to specify a `<toolbarComboBox>` element with a **name** attribute containing the string *Style*. Each `<comboBoxItem>` child element in `<toolbarComboBox>` will define the tag types the user can insert.
- The actual CSS loaded into EditLive! uses the methods outlined in [Setting Styles in EditLive! for Java Swing](#).

### Example

A stylesheet loaded into EditLive! for Java Swing defines the following styles:

```
h1{color: yellow}
p.blue{color: blue}
span.red{color: red}
.green{color: green}
```

The EditLive! for Java Swing Configuration File used features the following `<toolbarComboBox>` element:

```
<toolbarComboBox name="Style">
  <comboBoxItem name="P" text="Normal"/>
  <comboBoxItem name="H1" text="Heading 1"/>
</toolbarComboBox>
```

The following would appear in the drop-down styles list box:

Paragraph Styles	
Normal	¶
Heading 1	¶
Normal (blue)	¶
.green	¶
Inline Styles	
.red	Ⓐ
.green	Ⓐ

`.green` class appears twice as it can be applied as both an inline and a block style.

<comboBoxItem> Configuration File elements can use the text attribute to create an alias for the Style type. In the example above, you can see that H1 tags appear as Heading 1 instead of H1. For more information on creating aliases for styles, see the <comboBoxItem> article.

## Styles Drop-Down Hierarchy

The styles drop down displays available styles according to where the cursor has been inserted. It will display styles in the following order:

- **Paragraph Styles** (e.g. P tag, Heading tags)
- **Inline Styles** (e.g. Span tags )
- **Element Styles** (styles available to the current element selected)
- **Parent Element Styles** (styles for each parent element of the currently selected element).

### Example

The following HTML is loaded into EditLive! for Java Swing:

```
<h1>Table Example</h1>
<table>
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
```

If the user selected the text *Cell 1*, the style drop-down would display the following available styles:

- Paragraph Styles
- Inline Styles
- Cell Styles
- Row Styles
- Table Styles

Styles will appear under their defined type (e.g. paragraph, inline, element) in the following hierarchy:

- Default element styles (e.g. h1)
- All element specific CSS class (e.g. h1.h1Style)
- All generic CSS classes (e.g. .genericStyle)

The order by which default element styles, element specific styles, and generic styles are arranged will depend on the order in which these styles appear when passed to EditLive! for Java Swing.

### Example

A stylesheet loaded into EditLive! for Java Swing defines the following styles:

```
p.blue{color: blue}
p.yellow{color: yellow}
.green{color: green}
.brown{color: brown}
```

The Styles drop-down will first render the default element styles (i.e. *p.blue*, then *p.yellow*). The generic elements will then be rendered in the order *.green* then *.brown*.

### Example

A stylesheet loaded into EditLive! for Java Swing defines the following styles:

```
h1{color: yellow}
p.blue{color: blue}
span.red{color: red}
.green{color: green}
```

The EditLive! for Java Swing [Configuration File](#) features the following <toolbarComboBox> element:

```
<toolbarComboBox name="Style">
  <comboBoxItem name="P" text="Normal"/>
  <comboBoxItem name="H1" text="Heading 1"/>
</toolbarComboBox>
```

The following would appear in the drop-down styles list box:

Paragraph Styles	
Normal	¶
Heading 1	¶
Normal (blue)	¶
.green	¶
Inline Styles	
.red	a
.green	a

You can see that the default styles for the <P> and <H1> tags appear first. Then CSS class specific to <P> (*.blue*) appears next. Finally, the generic CSS class *.green* appears last.

## Getting Styles from EditLive! for Java

Style information for EditLive! for Java can be extracted using the following methods:

- [getStyles\(\)](#) method for an instance of the ELJBean class.
- [getDocument\(\)](#) method for an instance of the ELJBean class.  
This method returns the entire XHTML Document, including the embedded styles in the <head> tag.

## See Also

- [Retrieving Content From EditLive! for Java Swing](#)
- [<head>](#) Configuration File Element
- [<link>](#) Configuration File Element
- [<wordImport>](#) Configuration File Element

# Using CSS Extensions to Render Custom Tags

EditLive! includes extensive support for the embedding of custom tags within HTML. As part of the support for custom tags in HTML, developers can provide custom rendering for these tags. Tiny (Ephox) CSS extensions can be used to customize the rendering of these tags. The styles with which custom tag rendering is specified in EditLive! can be provided either as an external, linked style sheet or through an embedded style sheet. Please see the section on [Using CSS in EditLive!](#) for more information on using style sheets with EditLive!.

## Specifying Rendering for a Custom Tag

A CSS style can be used to specify the way in which a custom tag is to be rendered. The style should match the name of the custom tag. Developers can specify whether the custom tag is rendered as an inline (e.g. <SPAN>), block (e.g. <P>), or empty tag (e.g. <IMG>). Furthermore, icons and labels can be specified to be used for rendering the tag.

Inline style information should not be used to specify rendering of a custom tag.

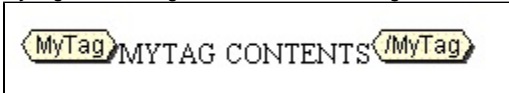
### Example

The following example demonstrates how to specify rendering information for the custom tag <MyTag> and its closing tag </MyTag>. The start icon starticon.jpg (mapped to by the URL <http://www.server.com/icons/starticon.jpg>), the end icon endicon.gif (mapped to by the URL <http://www.server.com/icons/endicon.gif>), the start label Custom Tag and end label /Custom Tag are specified for rendering the custom tag. Also, the custom tag is displayed as a block tag.

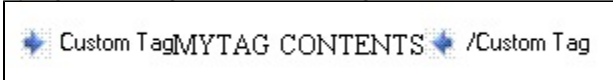
The display attribute must be assigned when specifying a custom tag to be used with EditLive!. The value of the display attribute affects the way in which the custom tag is processed by EditLive!.

```
MyTag{
  display: inline;
  ephox-start-icon: url(http://www.server.com/icons/starticon.jpg);
  ephox-end-icon: url(http://www.server.com/icons/endicon.jpg);
  ephox-start-label: Custom Tag;
  ephox-end-label: /Custom Tag;
}
```

MyTag Custom Tag With **No** CSS Rendering



MyTag Custom Tag **With** CSS Rendering



## CSS Extensions for Custom Tags

The Tiny (Ephox) CSS extensions can be implemented in either an external style sheet or an embedded element style sheet.

- display* CSS Attribute
- ephox-start-icon* CSS Attribute
- ephox-start-label* CSS Attribute
- ephox-end-icon* CSS Attribute
- ephox-end-label* CSS Attribute
- ephox-icon* CSS Attribute
- ephox-label* CSS Attribute

### display Attribute

#### Description

The display attribute specifies what type of rendering area should be allocated to the relevant tag. It also affects the way in which EditLive! interprets the tag when parsing the content.

#### Permitted Values

##### *block*

The tag and content are displayed. Custom block tags will not be automatically wrapped within other tags.

##### *inline*

The tag and content are displayed. The tag must contain content; otherwise, it will be removed. Custom inline tags will be automatically wrapped within a block tag (i.e. <P>).

##### *empty*

Only the tag is displayed. This display type should only be used with custom tags that do not contain content.

### Example

The following example specifies that the custom tag `<MyTag>` is to be rendered and interpreted as an inline tag and will be rendered with the label *Custom Tag*.

```
MyTag{
  display: inline;
  ephox-label: Custom Tag;
}
```

### ephox-start-icon Attribute

#### Description

This CSS attribute can be used to specify the icon used for an opening custom tag (e.g. `<CustomTag>`). This attribute should not be used with an empty tag. The icon for empty tags can be specified through the *ephox-icon* attribute.

#### Permitted Values

*url*  
A URL which maps to an image which is to be used as the icon for this custom tag. This URL can be either relative or absolute. If relative URLs are specified they are resolved in relation to either the BaseURL load-time property (if specified) or the page in which EditLive! is embedded (if the BaseURL load-time property is not specified).

### Example

The following example specifies the image *icons/starticon.gif* for the start tag icon and *icons/endicon.gif* for the end tag icon of the custom tag `<MyTag>`.

```
MyTag{
  display: block;
  ephox-start-icon: url(icons/starticon.gif);
  ephox-end-icon: url(icons/endicon.gif);
}
```

### ephox-start-label Attribute

#### Description

This CSS attribute can be used to specify the label used for an opening custom tag (e.g. `<CustomTag>`). This attribute should not be used with an empty tag. The label for empty tags can be specified through the *ephox-label* attribute.

#### Permitted Values

*label*  
A label to be used with the opening tag of a custom tag.

### Example

The following example specifies the label *Custom Tag* for the start tag label and */Custom Tag* for the end tag label of the custom tag `<MyTag>`.

```
MyTag{
  display: block;
  ephox-start-label: Custom Tag;
  ephox-end-label: /Custom Tag;
}
```

### ephox-end-icon Attribute

#### Description

This CSS attribute can be used to specify the icon used for a closing custom tag (e.g. `</CustomTag>`). This attribute should not be used with an empty tag. The icon for empty tags can be specified through the *ephox-icon* attribute.

#### Permitted Values

*url*  
A URL which maps to an image which is to be used as the icon for this custom tag. This URL can be either relative or absolute. Relative URLs are resolved in relation to either the BaseURL load-time property (if specified) or the page in which EditLive! for Java is embedded (if BaseURL is not specified).

### Example

The following example specifies the image *icons /starticon.gif* for the start tag icon and *icons /endicon.gif* for the end tag icon of the custom tag `<MyTag>`.

```
MyTag{
```

```
display: block;
ephox-start-icon: url(icons/starticon.gif);
ephox-end-icon: url(icons/endicon.gif);
}
```

## ephox-end-label Attribute

### Description

This CSS attribute can be used to specify the label used for a closing custom tag (e.g. </CustomTag>). This attribute should not be used with an empty tag. The label for empty tags can be specified through the *ephox-label* attribute.

### Permitted Values

*label*

A label to be used with the end of a custom tag.

### Example

The following example specifies the label *Custom Tag* for the start tag label and */Custom Tag* for the end tag label of the custom tag <MyTag>.

```
MyTag{
  display: block;
  ephox-start-label: Custom Tag;
  ephox-end-label: /Custom Tag;
}
```

## ephox-icon Attribute

### Description

This CSS attribute can be used to specify the icon used for a custom tag. If using this attribute with either block or inline tags then it should be used instead of the *ephox-start-icon* and *ephox-end-icon* attributes. When specifying an icon to represent an empty tag this attribute must be used.

### Permitted Values

*url*

A URL which maps to an image which is to be used as the icon for this custom tag. This URL can be either relative or absolute. Relative URLs are resolved in relation to either the BaseURL load-time property (if specified) or the page in which EditLive! for Java is embedded (if BaseURL is not specified).

### Example

The following example specifies the image *icons/icon.gif* for the start and end tag icons of the custom tag <MyTag>.

```
MyTag{
  display: block;
  ephox-icon: url(icons/icon.gif);
}
```

## ephox-label Attribute

### Description

This CSS attribute can be used to specify the label used for a custom tag. If using this attribute with either block or inline tags then it should be used instead of the *ephox-start-label* and *ephox-end-label* attributes. This attribute *must* be used when specifying an label to represent an empty tag.

### Permitted Values

*label*

A label to be used with the start and end tags of a custom tag.

### Example

The following example specifies the label *Custom Tag* for the start and end tag labels of the custom tag <MyTag>.

```
MyTag{
  display: block;
  ephox-label: Custom Tag;
}
```

## See Also

- [Using CSS in EditLive!](#)



# Restricting what CSS classes appear in EditLive! styles drop down

When specifying CSS for use in an instance of EditLive!, you may not necessarily want every style to appear in the styles drop-down combo box. Using specific CSS attributes, you can hide or display specified CSS elements in the styles drop-down in EditLive!.

## Showing or Hiding the Style

Using the *stylesVisibility* attribute of the `<wysiwygEditor>` configuration parameter you can limit the styles that are displayed such that

- only styles that **have the CSS attribute** *ephox-visible* set to "true" are shown. This is known as "whitelisting", or
- all styles are displayed **except** styles with the *ephox-visible* set to "false". This is known as "blacklisting"

the default behaviour of EditLive! is to use blacklisting.

## Identify Specific Styles

To show/hide a style in the styles drop-down combo box in EditLive!, developers can use the *ephox-visible* CSS attribute.

If the attribute is not included, the decision on if it's included or not is dependent on the *stylesVisibility* attribute of the `<wysiwygEditor>` configuration parameter

### ephox-visible Attribute

#### Description

The *ephox-visible* attribute specifies whether the style this attribute belongs to will render in the Styles drop-down combo box in EditLive!.

#### Permitted Values

*true*

The style this attribute belongs to *will* appear in EditLive!'s style drop-down combo box.

*false*

The style this attribute belongs to will *not* appear in EditLive!'s style drop-down combo box.

## Example

The following style sheet has three styles showing different values of the *ephox-visible* attribute.

```
.myCSSClass {
}
.navigation {
  ephox-visible: false;
}
.content {
  ephox-visible: true;
}
```

If the `<wysiwygEditor>` attribute *stylesVisibility* is not included, or set to "blacklist" the following styles will appear in in EditLive!'s style drop-down combo box.

- .myCSSClass
- .content

If the `<wysiwygEditor>` attribute *stylesVisibility* is set to "whitelist" the following styles will appear in in EditLive!'s style drop-down combo box.

- .content

## See Also

- [Using CSS in EditLive!](#)
- `<wysiwygEditor>`



# Proofing and Language Tools

- [Tiny Dictionaries](#)
- [Creating, Modifying and Adding to Dictionaries](#)
- [Tiny Thesauruses](#)
- [Auto Correct Spelling](#)

# Tiny Dictionaries

EditLive! comes packaged with several language-specific dictionaries. These dictionaries are used in conjunction with the EditLive! spell-checking functionality.

The `<spellCheck (Applet)>` configuration file element is used to specify which dictionary is used with a particular instance of EditLive!.

## Available Dictionaries

All dictionaries are packaged in the `redistributables/editlivejava/dictionaries` directory bundled with this SDK.

Dictionary Name	Language
en_us_4_0.jar	English (US)
en_us_medical_4_0.jar	English (US) with additional medical terms
en_combined.jar	Combined English (US) and English (UK)
en_br_4_0.jar	English (UK)
en_br_medical_4_0.jar	English (UK) with additional medical terms
pb_4_0.jar	Brazilian Portuguese
da_4_0.jar	Danish
du_4_0.jar	Dutch
fi_4_0.jar	Finnish
fr_4_0.jar	French - European and Canadian
ge_4_0.jar	German
it_4_0.jar	Italian
no_4_0.jar	Norwegian
po_4_0.jar	Portuguese (Iberian)
sp_4_0.jar	Spanish - European, Mexican and South American
sw_4_0.jar	Swedish

## See Also

- [<spellCheck \(Applet\)> Configuration File Element](#)
- [Creating, Modifying and Adding to Dictionaries](#)

# Creating, Modifying and Adding to Dictionaries

Custom dictionaries may be created for use with Tiny EditLive!. Custom dictionaries are used to add words to an existing dictionary used by the EditLive! spell checker. This document details how to extend the existing dictionary of the EditLive! spell checker. This requires some skill with unzipping files and using the Java `.jar` command.

## Specifying a Custom Dictionary

1. In order to add a custom dictionary to an existing spell checker, the existing spell checker `.jar` file must first be unzipped. This file can be found in the *web folder/redistributables/editlivejava* directory of your EditLive! install. The name of this `.jar` file will vary with the spell checker your version of EditLive! is using. If in doubt as to the name of your existing dictionary, please see the [<spellCheck \(Applet\)>](#) configuration file element.

For the purposes of this document and the examples contained herein, it will be assumed that the spell checker concerned is named *en\_us\_4\_0.jar*.

Once you have found this `.jar` file, use a file unzipping utility to extract the contents of the `.jar`.

2. If the file has been extracted correctly the contents should contain 2 directories: *com* and *META-INF*.

If a custom dictionary has already been created for this spell checker, there will also be a file named *userdic.tlx* present. If this is the case and you wish to extend the existing custom dictionary then *do not* delete this file. Please see the Modifying the Current Custom Dictionary section of this document for more information.

If, however, you wish to remove the current custom dictionary, then delete this file. For more information on this then please see Removing the Current Custom Dictionary section of this document.

3. With a plain text editor, create a file called *userdic.tlx* in the same directory where the contents of the `.jar` file were extracted. This file will be where the listings for the custom dictionary are placed.

4. At the top of the *userdic.tlx* file, place the following line:

```
#LID 24941
```

5. For each word that you wish to place into the custom dictionary, list the word in the *userdic.tlx* file on its own line. Then, one tab spacing from the word, place an *i* character. Thus, lines in the file should appear in the following format:

```
customword      i
```

6. Repeat step 5 for all the words you wish to add to the custom dictionary. When finished, save the *userdic.tlx* file.

7. After saving the *userdic.tlx* file the `.jar` file must be recompiled. This requires using the Java `.jar` command at the command line in the directory where the contents of the original `.jar` file were extracted.

### Example

If the contents of the dictionary `.jar` file were extracted to *c:\customdictionary* and the location of the `jar` command is *c:\java\bin\jar*, then the following command would create a `.jar` file called *customdictionary.jar*.

```
c:\customdictionary>c:\java\bin\jar cvf customdictionary.jar .
```

8. Move the newly compiled `.jar` file to a location where it may be accessed by the EditLive! applet.

9. Edit the configuration information for EditLive! to reflect the location of the new spell checker. See the [<spellCheck \(Applet\)>](#) configuration file element for more information on how to do this.

## Modifying the Current Custom Dictionary

1. Perform steps 1 and 2 as above in the Specifying a Custom Dictionary section of this document.

2. Open the *userdic.tlx* file in a plain text editor.

3. For each word that you wish to add to the custom dictionary list the word in the *userdic.tlx* file on its own line. Then, one tab spacing from the word, place an *i*. Thus, lines in the file should appear in the following format:

```
customword      i
```

4. Repeat step 3 for all the words you wish to add to the custom dictionary. When finished save the *userdic.tlx* file.

5. After saving the *userdic.tlx* file the `.jar` file must be recompiled. This requires using the Java `jar` command at the command line in the directory where the contents of the original `.jar` file were extracted. The name of the new spell checker `.jar` file is specified in this step.

### Example

If the contents of the dictionary .jar file were extracted to *c:\customdictionary* and the location of the jar command is *c:\java\bin\jar*, then the following command would create a jar file called *customdictionary.jar*.

```
c:\customdictionary>c:\java\bin\jar cvf customdictionary.jar .
```

6. Move the newly compiled .jar file to a location where it may be accessed by the EditLive!

7. Edit the configuration information for EditLive! to reflect the location of the new spell checker. See the [<spellCheck \(Applet\)>](#) configuration file element for more information on how to do this.

## Removing the Current Custom Dictionary

1. Perform steps 1 and 2 as above in the Specifying a Custom Dictionary section of this document.

2. Delete the *userdic.tlx* file from the directory where the contents of the original .jar file were extracted.

3. After deleting the *userdic.tlx* file the .jar file must be recompiled. This requires using the Java jar command at the command line in the directory where the contents of the original .jar file were extracted. The name of the new spell checker .jar file is specified in this step.

### Example

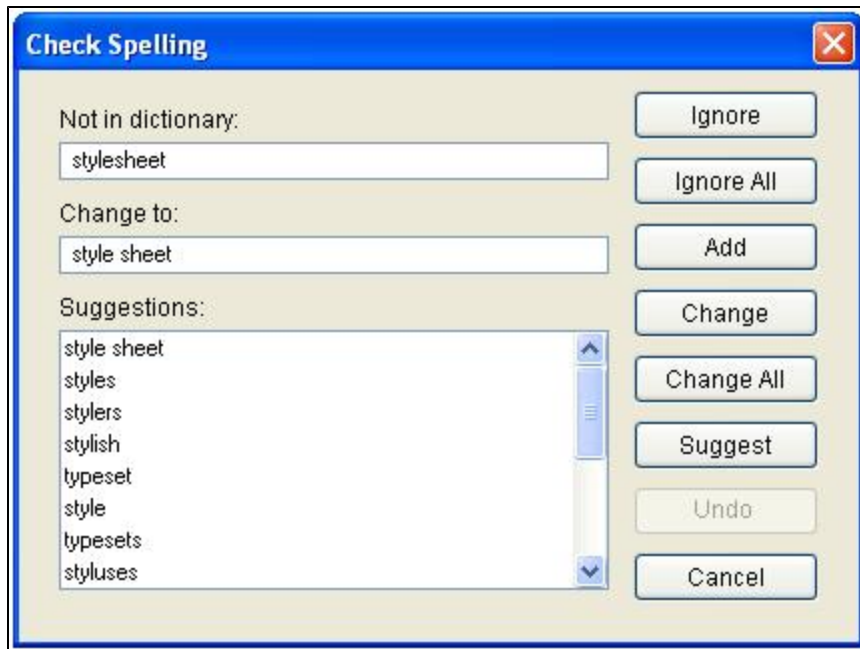
If the contents of the dictionary .jar file were extracted to *c:\customdictionary*, and the location of the jar command is *c:\java\bin\jar*, then the following command would create a .jar file called *customdictionary.jar*.

```
c:\customdictionary>c:\java\bin\jar cvf customdictionary.jar .
```

4. Move the newly compiled .jar file to a location where it may be accessed by the EditLive! applet.

## Adding Words to the Local Dictionary

Users can also add words to the dictionary which are then stored locally on the client. Any words added to the local dictionary by users will persist on the client even when the dictionary is updated on the server. Words are added to the local dictionary when a user clicks the Add Word option on the spell checker.



See Also

- [<spellCheck \(Applet\)>](#) Configuration File Element
- [Tiny Dictionaries](#)

# Tiny Thesauruses

EditLive! comes packaged with several language-specific thesauruses. These thesauruses allow the user to check a selected word for synonyms.

The [<thesaurus \(Applet\)>](#) configuration file element is used to specify which thesaurus is used with a particular instance of EditLive!.

## Available Thesauruses

All [thesauruses](#) are packaged in the *redistributables/editlivejava/thesaurus* directory bundled with this SDK.

Thesaurus Name	Language
thes_am_6_0.jar	English (US)
thes_br_6_0.jar	English (UK)
thes_ca_6_0.jar	English (Canadian)

## See Also

- [<thesaurus \(Applet\)>](#) Configuration File Element
- [Enabling Enterprise Edition](#)

# Auto Correct Spelling

EditLive! provides functionality for automatically correcting the spelling of certain words.

EditLive! currently only supports Auto Correct word lists for British and US English. You can generate custom auto correct word lists for additional languages using the format specified below.

## Enabling Auto Correct

To enable Auto Correct, you will need to set the **startAutoCorrect** attribute for the `<spellCheck (Applet)>` configuration file element to `true`. Auto Correct can be toggled by the user if you add the AutoCorrect menu item to the EditLive! interface.

## Modifying the Auto Correct Word List

The word list used by Auto Correct can be modified in the same way as the Tiny dictionary word lists. Customizations are performed in a text file packaged with a custom `dictionary.jar` file.

### Auto Correct Word List Format

The Auto Correct word list is stored in a file called `correct.tlx`, which is packaged inside your specified `dictionary.jar` file. The first line of this file must contain the exact text:

```
#LID 24941
```

Each preceding line in the file will depict an incorrect word and its correctly spelled replacement. These lines must adhere to the following format:

```
word \t action otherword
```

- **word**  
The incorrectly spelled word to be detected by the editor.
- **\t**  
The tab character.
- **action**  
The action to be invoked against the specified word. The actions available are:
  - **a**  
The specified word will be automatically be changed on detection.
  - **A**  
The specified word will be automatically changed, preserving the case pattern of the word as written in the document.
  - **c**  
This action currently works the same as the a action.
  - **C**  
This action currently works the same as the A action.

## Example

The following example specified an Auto Correct word list that will change the word `teh` to the word `the`, preserving the case.

```
#LID 24941  
teh      Athe
```

## See Also

- `<spellCheck (Applet)>` Configuration File Element
- [Tiny Dictionaries](#)

# Image and Media Support

EditLive! supports working with a wide variety of media from images to interactive maps and social media. This section of the documentation provides an overview of how to take advantage of this functionality.

## Working with Images

EditLive! includes rich functionality for working with images. It enables the insertion of images from both web servers and local machines. With EditLive! Enterprise Edition users can also resize, resample and edit images within the editor, without any extra desktop tooling.

For more information see the [Working with Images](#) document.

## Working with Media

With EditLive! users can also embed a wide variety of media assets within their content. EditLive! supports embedding media in multiple ways. This includes the following:

- Linking to content on public media services
- Using HTML5 AUDIO and VIDEO tags to link to web-based media
- Embedding content using IFRAME tags

For more information see the [Working with Media](#) document.

## Advanced Media Support

EditLive! also supports several advanced media integration techniques including working with <object> tags.

For more information on how to use these capabilities see the [Advanced Media Support](#) documentation.

# Working with Images

EditLive! supports the addition of images into content from both the web and the local machines of users. Images can be added to content via EditLive!'s Insert Image dialog, via a copy and paste operation or via drag and drop.

## Image Editing

Once an image has been inserted into EditLive! it can also be edited with the built in [Image Editor](#) - a feature of [EditLive! Enterprise Edition](#). EditLive!'s built in image editor enables basic images, resamples images when they're resized (resulting in smaller file sizes) and enables EditLive! to be configured to limit the size (in pixels) of any local image added to the document.


EditLive!'s built in image editor provides the following functionality:

- Automatic resize and resampling of images to fit within specified size constraints.
- Automatic resampling of images when they are resized within EditLive!
- Crop images
- Flip image vertically and horizontally
- Rotate images
- Apply basic image effects

## The Benefits of Enabling Image Upload

It is important to ensure that you enable image upload for local images with EditLive!. If image upload has not been enabled or is not correctly configured then local images will not be uploaded and appear as broken images within EditLive!'s content.

How Local Images are Added to EditLive!

 Local images can be added to content in EditLive! by the following means:

- Insertion via the Insert Image dialog
- Insertion as a background image (e.g. in the table properties dialog)
- Copy and paste from an external application (e.g. Microsoft Word)
- Drag and Drop of an image file
- Editing or resizing an image with EditLive!'s image editor

It is particularly important to enable local image upload to ensure that any images inserted into EditLive! via a copy and paste operation are preserved. When images are copied they are often stored as a local temporary file by the clipboard. Image upload enables EditLive! to upload these temporary files to the relevant server and adjust the URL to those files accordingly.

For more information on how to enable image upload in EditLive! see the documentation on [HTTP Upload Support for Images and Objects](#).

## Base64 Encoded Images

EditLive! 9 (version 9.0.2 and above) introduces support for the insertion of Base64 encoded images. Base64 encoding allows images to be embedded into EditLive! content, without the requirement that the image be uploaded to the server. Base64 encoded images can be added via inserting the data URI into the *Image URL* field in EditLive!'s Insert Image dialog, via a copy and paste operation or via drag and drop.

Editing operations will convert the Base64 encoded image to a regular image, requiring that the image be uploaded to preserve changes.



# Image Editing

The Image Editing feature of EditLive! allows users to modify images contained in the editor. Users can perform actions such as rotate, flip, and crop for any image.

In order for users to utilize the Image Editing functionality, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing, please see the [Licensing EditLive!](#) article.

## Enabling Image Editing

In order to enable Image Editing, the following steps will need to be performed.

### Ensure Local Images are Uploaded from EditLive!

When a user performs image editing, the modified image is stored on the user's local machine. Hence, images are required to be uploaded to a server location in order for images to be displayed for all users.

For information on specifying upload scripts and triggering image uploading, refer to the article [HTTP Upload Support for Images and Objects](#).

### Enabling the Image Editing Plugin

The Image Editor is a [plugin](#) packaged by default with EditLive!. Because this plugin is packaged by default, you can simply specify the *name* attribute **imageEditor** for a `<plugin>` element resident in your EditLive! [Configuration File](#).

#### Example

```
<editlive>
  ...
  <plugins>
    <plugin name="imageEditor" />
  </plugins>
</editlive>
```

### Specifying the Image Editing Toolbar and/or Menu Items

Numerous Image Editor specific toolbar/menu items are documented in the [Menu and Toolbar Item List](#). An inline toolbar can be applied for any image in EditLive! using the `<inlineToolbar>` configuration element.

## Limitations of Image Editing

### Transparency Effects on Images Resample Images as PNG Format

When a transparency effect is applied to an image (such as the *Reflection* or *Rounded Corners* effects), the image will be changed to the PNG file format.

### EditLive! is Required to Write to the User's Local File System

When edits are made to an image in EditLive!, the editor stores the new image on the user's file system. As noted above, these new local images must be uploaded to a server-side location to ensure all content observers can view the new image.

When EditLive! loads for the first time, users are prompted with a dialog asking if they wish to run the signed Java applet. This allows EditLive! to perform write actions to the user's file system. However, if the user's file system has been completely locked down using the operating system's security permissions, EditLive! will be unable to locally store edits made to images. For this reason, the user's file system needs to allow write permissions to applications.

### Large Images will Automatically be Rescaled

If images with a large resolution are inserted into the editor via the [Image Insertion](#) dialog, these images will automatically be rescaled to have a smaller resolution. This rescaling is performed to ensure images are kept at a usable size both to maximize usability and minimize memory consumption by the editor.

### Resized Images will Automatically be Rescaled

If the Image Editor is enabled, regardless of which toolbar buttons or menu items are enabled for image editing, resizing images will rescale the image and create a copy on the user's local file system.

## See Also

- `<images>` Configuration file element

# HTTP Upload Support for Images and Objects

## Overview

When inserting and uploading local images and objects (multimedia files) to a remote server, Ephox EditLive! uses the HTTP multipart form-data protocol.

The `<httpUpload>` configuration file element contains the configuration information for the HTTP Upload functionality for images and media embedded via `<object>` tags.

### Why Use HTTP?

Using the HTTP POST method to insert and upload images and objects offers a secure way of allowing end users to interact with the remote server that the images and objects are to be stored on.

## Requirements for HTTP Upload

In order to upload local files to the remote server via HTTP, you will need a server-side upload handler script that accepts the images and objects on the server and stores them in the correct directory or database. This script is the same script that would be used for uploading any file to the server via the HTTP POST method.

For example, when you use a file input element (i.e. `<INPUT type="file">`), the script specified in the action attribute of the parent `<form>` element is used to upload the file to the server.

```
<html>
  <body>
    ...
    <form method="POST" action="http://www.yourserver.com/scripts/upload.jsp">
      <input type="file" name="imageSelection">
    </form>
    ...
  </body>
</html>
```

This script would be the same script specified through EditLive! to upload images to the remote server.

## EditLive! HTTP Upload Configuration

The EditLive! HTTP upload configuration requires a URL corresponding to the image upload handler script as well as a base property that will be used to replace all existing local image and object URLs after the upload. These properties can be set via the `href` and `base` attributes of the `<httpUpload>` element respectively.

### Example

For the following configuration file example, the image upload script is located at `http://www.yourserver.com/scripts/upload.jsp`. The uploaded files can be found in a directory with the Absolute URL `http://www.yourserver.com/userfiles/`. (i.e. all local images and objects URLs will be replaced with the URL `http://www.yourserver.com/userfiles/`.)

```
<editLive>
  ...
  <mediaSettings>
    ...
    <httpUpload
      base="http://www.yourserver.com/userfiles/"
      href="http://www.yourserver.com/scripts/upload.jsp" />
    ...
  </mediaSettings>
  ...
</editLive>
```

### Example

For the following example, the URL of the directory containing pages with EditLive! embedded in them is `http://yourserver.com/editlive/pages`. The URL of the directory containing images is `http://yourserver.com/editlive/repository/images`. Hence, the relative base URL for displaying images in EditLive! would be `../repository/images`.

The upload handler script is located at `http://yourserver.com/scripts/uploadhandler.asp`

```

<editLive>
  ...
  <mediaSettings>
    ...
    <httpUpload
      base=" ../repository/images"
      href="http://yourserver.com/scripts/uploadhandler.asp" />
    ...
  </mediaSettings>
  ...
</editLive>

```

Hence, if an image *myImage.jpg* is located at *http://yourserver.com/editlive/repository/images* and a HTML page located at *http://yourserver.com/editlive/pages/myPage.html* wants to reference this image, the URL would be *../repository/images/myImage.jpg*. With the above configuration file settings, this path would be correct.

Ephox recommends using absolute URLs when possible, as it is less likely to lead to confusion.

### When are Files Actually Uploaded?

Images and embedded object files are uploaded (via your upload script) in the following circumstances:

- EditLive! is embedded in a HTML <form>, which is then submitted via a submit button.
- The [uploadImages Method](#) of EditLive! is invoked.
- The [getBody Method](#) of EditLive! is invoked, passing true as the second parameter.
- The [getDocument Method](#) of EditLive! is invoked, passing true as the second parameter.

When relying on images being uploaded when the containing <form> is submitted then the [setAutoSubmit Method](#) must **not** be set to false if local images are to be uploaded.

#### Example

```

<form name="form1" method="POST" action="http://www.yourserver.com/scripts/upload.jsp">
  <script src="../../redistributables/editlivejava/editlivejava.js"></script>

  <script language="JavaScript">
    var editlivejava1;
    editlivejava1 = new EditLiveJava("ELApplet1", "700", "400");
    ...
    editlivejava1.show();
  </script>
  <input type="submit" value="Submit" />
</form>

```

### What Form Fields are Files Uploaded In?

Local files which are uploaded to the server by EditLive! are placed within a form field with the name *image*, or to the field specified with the *uploadFileFieldName* attribute. When implementing an upload acceptor script specifically to receive files from EditLive! the script should be made to accept files submitted within the image field.

### Dynamically Setting the Image or Object URL

If you don't want every local image specified to have its URL changed to the **base** attribute specified, it is possible to dynamically assign the URL for images or objects embedded within the content as they are uploaded. This functionality is achieved via the POST acceptor for files. If the POST acceptor returns a string containing a URL, the URL is inserted into the document source code as the URL for the file. In this way it is possible to dynamically assign the directory files are uploaded to without having to dynamically generate the configuration file.

For this functionality to operate correctly the relevant upload acceptor script must only return a single line string with the URL corresponding to the location for the uploaded file.

When setting the file URL in this manner the URL returned by the POST acceptor script to EditLive! takes precedence over the **base** attribute of the [<httpUpload>](#) element in the EditLive! configuration file.

For examples of using a POST acceptor script to return a URL to an instance of EditLive!, see the [ASP.NET](#), [ASP](#), [Cold Fusion](#), [JSP](#), and [PHP](#) example Image Upload Scripts packaged with this SDK.

The URL returned by the POST acceptor must be exactly the URL to be used for the relevant file in EditLive!.

Ensure that the URL returned by the POST acceptor does **not** include a newline character. Note that newline characters can be introduced by using a method such as [println](#) or [writeln](#). Ensure you are using methods which do **not** introduce a newline character such as [print](#) or [write](#).

## Example Image Upload Scripts

EditLive! for Java is packaged with sample upload scripts for ASP, ASP.NET, JSP, PHP and ColdFusion that enable uploads for common image formats. The source for these scripts can be found in the *SDK\_INSTALL/webfolder/uploadscripts/* directory where *SDK\_INSTALL* represents the directory to where the EditLive! SDK is installed. The upload scripts can also be extended to be used with files associated with embedded objects (e.g. Macromedia Flash files). This is achieved by adding the relevant extensions to the list of known file types.

Documentation is also available for the upload acceptor scripts:

- [ASP Upload Script](#)
- [ASP.NET Upload Script](#)
- [JSP Upload Script](#)
- [PHP Upload Script](#)
- [Cold Fusion Upload Script](#)

### Common Problems

There are a number of problems that may occur while attempting to use HTTP Upload. These can be complicated and vary from server to server. For more information on the settings relevant for your server, please consult the documentation for your server. Some common problems include:

- Server side settings - ensure that the HTTP upload script exists in a directory in which scripts can be executed.
- File system permissions - ensure that the file permissions of the directory in which files are to be placed has write and read permissions set.

### See Also

- [<httpUpload> Configuration File Element](#)
- Example [ASP Upload Script](#)
- Example [ASP.NET Upload Script](#)
- Example [JSP Upload Script](#)
- Example [PHP Upload Script](#)
- Example [Cold Fusion Upload Script](#)

# ASP.NET Upload Script

ASP.NET allows for the easy creation of upload handler scripts. The following ASP.NET page allows files to be uploaded. The *fileupload.aspx* page does not process the upload; this task is performed by the *fileupload.aspx.cs* page.

The source for EditLive! multimedia upload scripts can also be found in the `SDK_INSTALL/webfolder/uploadscripts/` directory where `SDK_INSTALL` represents the directory to where the EditLive! SDK is installed.

## Defining the Location of the Image Upload Handler Script

The location of the image upload handler script must be defined within the EditLive! configuration file. This setting is configured via the `href` attribute of the `<httpUpload>` element of the configuration file. To use this example script, the `href` attribute should point to the location of this script on the server.

## Defining the Location of the Image Upload Directory

This example script uploads images to the directory specified by the *imageDir* variable. In order for images to function correctly within EditLive!, the `base` attribute of the `<httpUpload>` element must reflect the location of the directory where images are to be stored on the Web server.

## Example Image Upload Handler Script

Tiny has written a sample image upload handler script using Active Server Pages and VBScript. This script can be found at `SDK_INSTALL/webfolder/uploadscripts\aspnetfileUpload.aspx` and `SDK_INSTALL/webfolder/uploadscripts\aspnetfileUpload.aspx.cs`, where `SDK_INSTALL` represents the location where the EditLive! SDK is installed.

ASP.NET allows for the easy creation of upload handler scripts. The following ASP.NET page allows files to be uploaded. The *fileupload.aspx* page does not process the upload; this task is performed by the *fileupload.aspx.cs* page. The code for the *fileupload.aspx* page is as follows:

```
<%@ Page language="c#" Codebehind="fileupload.aspx.cs" AutoEventWireup="false" Inherits="Ephox.FileUpload" %>
```

The POST is then handled in the `Page_Load` method of the *fileupload.aspx.cs* page. This appears as follows:

```
private void Page_Load(object sender, System.EventArgs e)
{
    /*
     * Set the "path" variable to the location where images are to be stored
     */
    string path = "../images";

    HttpFileCollection files;
    files = Page.Request.Files;

    for(int index=0; index < files.AllKeys.Length; index++)
    {
        HttpPostedFile postedFile = files[index];
        string fileName = null;
        int lastPos = postedFile.FileName.LastIndexOf('\\');
        fileName = postedFile.FileName.Substring(++lastPos);

        //Check the file type through the extension
        if(fileName.EndsWith(".jpg") || fileName.EndsWith(".jpeg") ||
           fileName.EndsWith(".gif") || fileName.EndsWith(".png") ||
           fileName.EndsWith(".tiff"))
        {
            postedFile.SaveAs(MapPath(path + "/" + fileName));
        }
    }
}
```

## Configuring EditLive! to use the Image Upload Script

Below are the steps required to use the ASP.NET image upload handler from above with EditLive! in your own Web application.

1. One line of code in the *fileUpload.aspx.cs* file must be changed for image upload.
  - This line of code specifies the location where you wish image files to be uploaded to. If the location of the upload acceptor script was `http://www.yourserver.com/scripts/fileUpload.aspx`, then setting the path variable to `../images` would upload the images to a directory with the URL `http://www.yourserver.com/images/`.

```
string path = "../images";
```

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

2. EditLive!'s configuration file should now be edited to reflect the changes made in the previous step. You will find these settings within the `<httpUpload>` element. The URL setting should reflect the location of the `fileUpload.aspx` file on your Web server.
  - The following example reflects the setting of the `href` attribute of the `<httpUpload>` element if the upload script was at the URL `http://www.yourserver.com/scripts/fileUpload.aspx`.

```
<editLive>
...
<mediaSettings>
  <httpUpload
    base= ...
    href="http://www.yourserver.com/scripts/fileUpload.aspx" />
  ...
</mediaSettings>
...
</editLive>
```

3. Finally the HTTP Image Upload `base` attribute should be changed to reflect the location where images can be found on your Web server.

This location may not be the same value as that used within the upload acceptor script above. Rather, it will be the virtual directory alias used by your Web server for the location listed in the upload acceptor script.

- This example follows from the code above. It uses an absolute URL as the value of the `base` attribute. The value of the `base` attribute corresponds to the URL that for the directory that images are uploaded to.

```
<editLive>
...
<mediaSettings>
  <httpImageUpload
    base="http://www.yourserver.com/images/"
    href="http://www.yourserver.com/scripts/fileUpload.aspx" />
  ...
</mediaSettings>
...
</editLive>
```

## Extending the Image Upload Script for Use with Other File Types

The ASP.NET upload acceptor script provided with EditLive! can, by default, only be used with common image file types. This is restricted by inspecting the extension of an uploaded file. Support for other file types, including multimedia object types such as Macromedia Flash (.swf) and Apple QuickTime (.mov), can easily be added by adding the relevant extension to the list of allowed extensions in the upload `fileupload.aspx.cs` portion of the acceptor script.

The list of permitted extensions can be found as follows in the ASP.NET `fileupload.aspx.cs` upload acceptor script file in the following statement:

```
HttpFileCollection files;
files = Page.Request.Files;

for(int index=0; index < files.AllKeys.Length; index++)
{
  HttpPostedFile postedFile = files[index];
  string fileName = null;
  int lastPos = postedFile.FileName.LastIndexOf('\\');
  fileName = postedFile.FileName.Substring(++lastPos);

  //Check the file type through the extension
  if(fileName.EndsWith(".jpg") || fileName.EndsWith(".jpeg") ||
     fileName.EndsWith(".gif") || fileName.EndsWith(".png") ||
     fileName.EndsWith(".tiff"))
  {
    postedFile.SaveAs(MapPath(path + "/" + fileName));
  }
}
```

The example below demonstrates how this example can be extended to handle Flash and QuickTime files. Note the addition of the file extensions to the *if* statement in the code below:

```
HttpFileCollection files;
files = Page.Request.Files;

for(int index=0; index < files.AllKeys.Length; index++)
{
    HttpPostedFile postedFile = files[index];
    string fileName = null;
    int lastPos = postedFile.FileName.LastIndexOf('\\');
    fileName = postedFile.FileName.Substring(++lastPos);

    //Check the file type through the extension
    if(fileName.EndsWith("jpg") || fileName.EndsWith("jpeg") ||
        fileName.EndsWith("gif") || fileName.EndsWith("png") ||
        fileName.EndsWith("tiff") || fileName.EndsWith("swf") ||
        fileName.EndsWith("mov"))
    {
        postedFile.SaveAs(MapPath(path + "/" + fileName));
    }
}
```

## See Also

- [HTTP Upload Support for Images and Objects](#)

# ASP Upload Script

This article provides a sample script, written using Active Server Pages and VBScript, to upload images via the HTTP POST method. Instructions on how it can be tailored for use in your Web applications are also included. You will need to set this facility if you would like to be able to upload local images to the server.

Download the ASP Multimedia upload example code: [fileUpload.asp](#).

The source for EditLive multimedia upload scripts can also be found in the `SDK_INSTALL/webfolder/uploadscripts/` directory where `SDK_INSTALL` represents the directory to where the EditLive! SDK is installed.

## Defining the Location of the Image Upload Handler Script

The location of the image upload handler script must be defined within the EditLive! configuration file. This setting is configured via the `href` attribute of the `<httpUpload>` element of the configuration file. To use this example script, the `href` attribute should point to the location of this script on the server.

## Defining the Location of the Image Upload Directory

This example script uploads images to the directory specified by the `imageDir` variable. In order for images to function correctly within EditLive!, the `base` attribute of the `<httpUpload>` configuration file element must reflect the location of the directory where images on the Web server.

## Example Image Upload Handler Script

Tiny has written a sample image upload handler script using Active Server Pages and VBScript. This script can be found at `SDK_INSTALL/webfolder/uploadscripts\asp\fileUpload.asp`, where `SDK_INSTALL` represents the location where the EditLive! SDK is installed.

Below are the steps required to use the ASP image upload handler in your own Web application.

1. One line of code in the `fileUpload.asp` file must be changed for image upload.
  - This line of code specifies the location where you want to upload image files. If the location of the upload acceptor script was `http://www.yourserver.com/scripts/fileUpload.asp` then setting the `imageDir` variable to `../images` would upload the images to a directory with the URL `http://www.yourserver.com/images/`.

```
Dim imageDir
imageDir="../images"
```

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

2. The EditLive! configuration file should now be edited to reflect the changes made in the previous step. You will find these settings within the `<httpUpload>` configuration file element. The URL setting should reflect the location of the `fileUpload.asp` file on your Web server.
  - The following example reflects the setting of the `href` attribute of the `<httpUpload>` configuration file element if the upload script is at the URL `http://www.yourserver.com/scripts/fileUpload.asp`.

```
<editLive>
...
<mediaSettings>
  <httpUpload
    base=...
    href="http://www.yourserver.com/scripts/fileUpload.asp" />
  ...
</mediaSettings>
...
</editLive>
```

3. Finally, the HTTP Image Upload `base` attribute should be changed to reflect the location where images can be found on your Web server.

This location may not be the same value as that used within the upload acceptor script above. Rather, it will be the virtual directory alias used by your Web server for the location listed in the upload acceptor script.

- The following example reflects the setting of the `href` attribute of the `<httpUpload>` element if the upload script is at the URL `http://www.yourserver.com/scripts/fileUpload.asp`.

```
<editLive>
...
<mediaSettings>
  <httpUpload
    base="http://www.yourserver.com/images/"
    href="http://www.yourserver.com/scripts/fileUpload.asp" />
  </mediaSettings>
```



```
...  
</editLive>
```

## Extending the Image Upload Script for Use with Other File Types

The ASP upload acceptor script provided with EditLive! can, by default, only be used with common image file types. This is restricted by inspecting the extension of an uploaded file. Support for other file types, including multimedia object types such as Macromedia Flash (.swf) and Apple QuickTime (.mov), can easily be added by adding the relevant extension to the list of allowed extensions in the upload acceptor script.

The list of permitted extensions can be found at the bottom of the ASP upload acceptor script file in the following statement:

```
if OnlyExtention="jpeg" or _  
  OnlyExtention="jpg" or _  
  OnlyExtention="tiff" or _  
  OnlyExtention="png" or _  
  OnlyExtention="gif" then  
  call SaveFile(curDir & "\" & OnlyFileName, FileData)  
end if
```

The example below demonstrates how this example can be extended to handle Flash and QuickTime files:

```
if OnlyExtention="jpeg" or _  
  OnlyExtention="jpg" or _  
  OnlyExtention="tiff" or _  
  OnlyExtention="png" or _  
  OnlyExtention="gif" or _  
  OnlyExtention="swf" or _  
  OnlyExtention="mov" or _ then  
  call SaveFile(curDir & "\" & OnlyFileName, FileData)  
end if
```

See Also

- [HTTP Upload Support for Images and Objects](#)

# Cold Fusion Upload Script

This article provides a sample script, written using ColdFusion, to upload images via the HTTP POST method. Instructions on how it can be tailored for use in your Web applications are also included. You will need to set this facility if you would like to be able to upload local images to the server.

Download the Cold Fusion multimedia upload example code: [coldfusion\\_postacceptor.cfm](#).

The source for EditLive multimedia upload scripts can also be found in the `SDK_INSTALL/webfolder/uploadscripts/` directory where `SDK_INSTALL` represents the directory to where the EditLive! SDK is installed.

## Defining the Location of the Image Upload Handler Script

The location of the image upload handler script must be defined within the EditLive! configuration file. This setting is configured via the `href` attribute of the `<httpUpload>` element of the configuration file. To use this example script, the `href` attribute should point to the location of this script on the server.

## Defining the Location of the Image Upload Directory

This example script uploads images to the directory specified by the `imageFolder` variable. In order for images to function correctly within EditLive!, the base attribute of the `<httpUpload>` element must reflect the location of the directory where images on the Web server.

## Creating an Image Upload Handler Script

1. Use the `<CFSETTING>` tag to prevent extra whitespace from being written to the output. EditLive! uses all output returned for the saved filename; extra whitespace will cause image URLs to be incorrect. For this reason the `<CFSETTING>` tag must be the first thing in the document.

```
<CFSETTING EnableCFOutputOnly="Yes">
<!-- The above ensures only the <cfoutput> tag causes any output;
      EditLive! includes extra whitespace as part of the file name.
-->
```

2. Create a variable to store the location of the images on the server. In this example, we are using the `images` subfolder of the script location.

```
<!--*****
      * Change this line to set the upload folder *
      *****-->
<cfset uploadFolder="images">
```

3. Use the `expandPath` function to get the full path of the folder on the server.

```
<cfset imageFolder=expandPath("#uploadFolder#")>
```

4. Save the uploaded file to the `images` folder.

```
<!-- Save the uploaded file -->
<cffile action = "upload"
      fileField = "image"
      destination = "#imageFolder#"
      nameConflict = "MakeUnique">
```

5. Return the name of the file to ELJ in case the MakeUnique script directive caused the file to be renamed.

```
<!-- Notify ELJ of the filename -->
<cfoutput>#serverFile#</cfoutput>
```

## Example Image Upload Handler Script

Tiny has written a sample image upload handler script using ColdFusion. This script can be found at `SDK_INSTALL/webfolder/uploadscripts/coldfusion/coldfusion_postacceptor.cfm`, where `SDK_INSTALL` represents the location where the EditLive! SDK is installed.

For image upload, one line of code in the *coldfusion\_postacceptor.cfm* file must be changed. This line of code specifies the location where you wish image files to be uploaded to. If the location of the upload acceptor script was *http://www.yourserver.com/scripts/coldfusion\_postacceptor.cfm*, then setting the *imageDir* variable to *../images* would upload the images to a directory with the URL *http://www.yourserver.com/images/*.

```
$imageFolder = "images/";
```

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

## Integrating the Upload Script

1. Open your configuration file in any text editor (e.g. Notepad on Windows).
2. Locate the `<mediaSettings>` element and add a `<httpUpload>` element to it. Only a portion of the full `<mediaSettings>` element is shown here.

```
<mediaSettings>
  <httpUpload
    base="images/"
    href="coldfusion_postacceptor.cfm">
  </httpUpload>
  ...
</mediaSettings>
```

## See Also

- [HTTP Upload Support for Images and Objects](#)

# JSP Upload Script

This article provides a sample script, written using a Java servlet, to upload an image file via HTTP POST. Instructions on how it can be tailored for use in Web applications are also included.

Download the JSP multimedia upload example code: [jsp\\_upload\\_example.zip](#).

The source for EditLive multimedia upload scripts can also be found in the `SDK_INSTALL/webfolder/uploadscripts/` directory where `SDK_INSTALL` represents the directory to where the EditLive! SDK is installed.

## Defining the Location of the Upload Handler Script

The location of the image upload handler script must be defined within the EditLive! configuration file. This setting is configured via the `href` attribute of the `<httpUpload>` element of the configuration file. To use this example script, the `href` attribute should point to the location of this script on the server.

## Defining the Location of the Upload Directory

This example script uploads images to the directory specified by the `FILEDIR` property within the `FILEUPLOAD.properties` file. In order for images to function correctly within EditLive!, the base attribute of the `<httpUpload>` element must reflect the location of the directory where images are uploaded to on the Web server.

## Setting Up the Sample JSP Upload Script

The JSP Image Upload Script provided with EditLive! is dependant on the Apache Commons FileUpload and Logging packages. These packages are provided with the source for the JSP upload script in the `SDK_INSTALL/webfolder/uploadscripts\jsp\lib` directory, where `SDK_INSTALL` is the directory where the EditLive! for Java is installed. The source for the example script can be found in the `UploadScript.java` file.

An implementation of the J2EE servlet API is also required to use the same upload script. The `servlet-api.jar` file from the Apache Tomcat project has been included in the `SDK_INSTALL/webfolder/uploadscripts\jsp\lib` directory. If this is not the case, please change the example code according to the location where this file can be found on your machine.

1. For file upload, the `FILEDIR` property in the `FILEUPLOAD.properties` file must be changed to indicate the location to which files are to be uploaded. Changing the `FILEUPLOAD.properties` does **not** require `UploadScript.java` to be recompiled.

```
FILEDIR=C:\\webserver\\webapp\\images\\
```

2. Should you wish to alter the `UploadScript.java` file, it must be recompiled for the changes to take effect. Once you have completed your changes, save the file. The file must now be compiled into a `.class` file. To compile this file you will need to run the Java compiler.
  - To run the Java compiler use the `javac` command from the command line. This command takes the form of:

```
javac -classpath <files needed for compilation> <file for compilation>
```

- Files that are needed for compilation should be listed with a semicolon ( ; ) separating them. The compilation of the `UploadTest.java` file requires the Apache Commons FileUpload library, as well as the servlet library (`servlet.jar`). Any spaces present in either the classpath or filename arguments should be enclosed by quotation (") marks as shown below. For example, the following would be the command line command which follows from the previous steps:

```
javac -classpath ".;C:\SDK_INSTALL\webfolder\uploadscripts\jsp\lib\commons-fileupload-1.0.jar; C:\SDK_INSTALL\webfolder\uploadscripts\jsp\lib\servlet-api.jar; C:\SDK_INSTALL\webfolder\uploadscripts\jsp\lib\commons-logging.jar" "C:\SDK_INSTALL\webfolder\uploadscripts\jsp\UploadScript.java"
```

These locations will change dependant on your install of the EditLive! J2EE SDK and your Web server install. The location of the `servlet-api.jar` file will depend on which Web server you are using. The `servlet-api.jar` file may be replaced by an appropriate equivalent for your Web server.

3. After this file has been correctly compiled (ie. the Java compiler has generated no errors and the `UploadScript.class` file has been generated), the `UploadScript.class` file must be copied to the `SDK_INSTALL/WEB-INF/classes` directory (where `SDK_INSTALL` represents the location that the EditLive! SDK install used by your Web server can be found). Following from the previous examples, the correct location for file would be:

```
c:\webserver\webapp\WEB-INF\classes\UploadScript.class
```

## Installing the Upload Script on the Web Server

Once the `UploadScript.class` has been copied to the `SDK_INSTALL/WEB-INF/classes` directory, the application's `web.xml` file must include an appropriate servlet mapping. The following steps detail how to do this:

1. Declare the servlet alias for the *UploadScript* class. The following XML declares *uploadScript* servlet alias for the *UploadScript* class. The `<servlet-name>` element contains the servlet alias, while the `<servlet-class>` tag contains the name of the class (found within the *WEB-INF/lib* directory of the Web application) to be used for the servlet.

```

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    <servlet-name>
      uploadScript
    </servlet-name>
    <servlet-class>
      UploadScript
    </servlet-class>
  </servlet>
  ...

```

2. Map a URL pattern to the servlet. The servlet will then be used to process HTTP requests corresponding to the URL pattern. The context for the URL pattern is the current Web application. Thus, a mapping of */uploadscript* within the application *editlivejava* would process all requests to the *http://webserver/editlivejava/uploadscript* URL where *webserver* represents the host server.
  - The following example maps the *uploadScript* servlet (as specified above) to the */uploadscript* URL pattern. The `<servlet-name>` element contains the servlet alias name and the `<url-pattern>` contains the URL pattern to be used to map to that servlet.

```

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    ...
  </servlet>
  <servlet-mapping>
    <servlet-name>
      uploadScript
    </servlet-name>
    <url-pattern>
      /uploadscript
    </url-pattern>
  </servlet-mapping>
  ...
</webapp>

```

## Configuring EditLive! to Use the JSP Image Upload Script

Below are the steps required to use the JSP image upload handler with EditLive! in your own Web application.

1. EditLive!'s configuration file should now be edited to reflect the changes made in the previous section. You will find these settings within the `<httpUpload>` element. The URL setting should reflect the location of the *UploadScript.class* file on your Web server.
  - The following example reflects the setting of the `href` attribute of the `<httpUpload>` element if the upload script could be found at the URL *http://www.yourserver.com/webapp/uploadscript*. See the previous section for information on how to configure the URL mapping for the upload script.

```

<editLive>
  ...
  <mediaSettings>
    <httpUpload
      base= ...
      href="http://www.yourserver.com/webapp/uploadscript" />
    ...
  </mediaSettings>
  ...
</editLive>

```

2. The HTTP Image Upload **base** attribute should be changed to reflect the location where images can be found on your Web server.



This location may not be the same value as that used within the upload acceptor script, above. Rather, it will be the virtual directory alias used by your Web server for the location listed in upload acceptor script.

- This example follows from the code above. It uses an absolute URL as the value of the **base** attribute. The value of the **base** attribute corresponds to the URL that for the directory that images are uploaded to.

```
<editLive>
...
<mediaSettings>
  <httpUpload
    base="http://www.yourserver.com/webapp/images/"
    href="http://www.yourserver.com/webapp/uploadscript" />
  ...
</mediaSettings>
...
</editLive>
```

## Extending the Image Upload Script for Use with Other File Types

The Java (JSP) upload acceptor script provided with EditLive! can, by default, only be used with common image file types. This is restricted by inspecting the extension of an uploaded file. Support for other file types, including multimedia object types such as Macromedia Flash (.swf) and Apple QuickTime (.mov), can easily be added by adding the relevant extension to the list of allowed extensions in the acceptor script's *FILEUPLOAD.properties* file.

To add the required extensions, simply edit the FILEEXT property of the *FILEUPLOAD.properties* file. This property is a comma-delimited list of extensions. Once you have specified the required extension, it is recommended that the server be restarted so that the change may take effect.

The list of permitted extensions can be found as follows in the *FILEUPLOAD.properties* file:

```
FILEEXT=jpg, jpeg, tiff, png, gif
```

The example below demonstrates how this example can be extended to handle Flash and QuickTime files. Note the addition of the file extensions to the property declaration below:

```
FILEEXT=jpg, jpeg, tiff, png, gif, swf, mov
```

See Also

- [HTTP Upload Support for Images and Objects](#)

# PHP Upload Script

This article provides a sample script, written using PHP, to upload images via the HTTP POST method. Instructions on how it can be tailored for use in your Web applications are also included. You will need to set this facility if you would like to be able to upload local images to the server.

Download the PHP multimedia upload example code: [php\\_postacceptor.php](#).

The source for EditLive! multimedia upload scripts can also be found in the `SDK_INSTALL/webfolder/uploadscripts/` directory where `SDK_INSTALL` represents the directory to where the EditLive! SDK is installed.

## Defining the Location of the Image Upload Handler Script

The location of the image upload handler script must be defined within the EditLive! configuration file. This setting is configured via the `href` attribute of the `<httpUpload>` element of the configuration file. To use this example script, the `href` attribute should point to the location of this script on the server.

## Defining the Location of the Image Upload Directory

This example script uploads images to the directory specified by the `imageFolder` variable. In order for images to function correctly within EditLive!, the `base` attribute of the `<httpUpload>` element must reflect the location of the directory where images are located on the Web server.

## Creating an Image Upload Handler Script

1. Create a variable to store the location of the images on the server. In this example, we are using the `images` subfolder of the script location.

```
<?
/*****
 * Change this line to set the upload folder *
 *****/
$imageFolder = "images/";
```

2. Reset the `_FILES` array, and store the first element in the variable `temp`.

```
reset ($_FILES);
$temp = current($_FILES);
```

3. Ensure the variable refers to a successfully uploaded file, and then save it to the folder set in step 1.

```
if (is_uploaded_file($temp['tmp_name'])){
    $filetowrite = $imageFolder . $temp['name'];
    move_uploaded_file($temp['tmp_name'], $filetowrite);
}
```

4. If the variable is not an uploaded file, return as error.

```
} else {
    // Notify EditLive! that the upload failed
    header("HTTP/1.0 500 Server Error");
}
?>
```

## Example Image Upload Handler Script

Tiny has written a sample image upload handler script using PHP. This script can be found at `SDK_INSTALL/webfolder/uploadscripts/php/php_postacceptor.php`, where `SDK_INSTALL` represents the location where the EditLive! SDK is installed.

For image upload, one line of code in the `php_postacceptor.php` file must be changed. This line of code specifies the location where you wish image files to be uploaded to. If the location of the upload acceptor script was `http://www.yourserver.com/scripts/php_postacceptor.php` then setting the `imageDir` variable to `../images` would upload the images to a directory with the URL `http://www.yourserver.com/images/`.

```
$imageFolder = "images/";
```

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

## Integrating the Upload Script

1. Open your configuration file in any text editor (e.g. Notepad on Windows).
2. Locate the `<mediaSettings>` element and add a `<httpUpload>` element to it.

Only a portion of the full `<mediaSettings>` element is shown here.

```
<mediaSettings>
  <httpUpload
    base="images/"
    href="php_postacceptor.php">
  </httpUpload>
  ...
</mediaSettings>
```

## See Also

- [HTTP Upload Support for Images and Objects](#)



# Drag and Drop

EditLive supports dragging and dropping of images into the editor from web pages and filesystem browsers such as Windows Explorer.

## Supported Platforms

Drag and drop support is only available on specific platforms as outlined below. Other platforms and scenarios may work but are not officially supported.

### Dragging from web pages

- Firefox and Chrome on Windows
- Firefox and Chrome on Ubuntu and Linux Mint

### Dragging from filesystem browser

- Firefox, Chrome and IE 9-11 on Windows
- Firefox and Chrome on Ubuntu and Linux Mint

Image drag and drop was introduced in EditLive 9.0.3

# Working with Media

EditLive! supports the embedding of media content through the use of oEmbed APIs. This section provides information on how the media support of EditLive! functions and how to configure media support within EditLive!.

## Media Overview

Media support within EditLive! is achieved by utilising oEmbed APIs to ask a service provider how one of their URLs should be embedded. This is described in the [oEmbed specification](#). EditLive! supports all four oEmbed types in the specification; photo, video, link and rich.



- The oEmbed "link" type is inserted as a plain hyperlink with no additional information. Editing these hyperlinks is done through the standard hyperlink dialog.
- The oEmbed "photo" type is inserted as an image with no additional information. These images are editable with the built in image editor and their properties can be adjusted using the Image Properties dialog.

Endpoints and URL scheme matching information must be provided for each service provider. This documentation is usually available from the service provider directly.

Media objects embedded within EditLive! are presented to the user in uneditable sections to preserve the output format returned by the service provider. Each section uses an overlay to indicate the type of oEmbed media that has been inserted. The icons can be seen below:



Used to represent rich, interactive and social media. Includes services such as SlideShare, Google Maps and Twitter.



Used to represent video media. Includes services such as YouTube, Vimeo and Brightcove

Photos inserted by media services, Flickr images for example, will be inserted as images.

To create an Insert Media menu item or toolbar button, the value *InsertMedia* must be specified for the **name** attribute of either a [<menuItem>](#) or [<toolbarButton>](#) configuration file element. For more information on EditLive! for Java configuration files, see the [Instantiating the Applet](#) article. For more information on creating menu or toolbar items, see the [Setting Menu and Toolbar Items](#) and [Menu and Toolbar Item List](#) articles.

## oEmbed fallback

Due to the huge number of oEmbed service providers and the differences between how each service provider returns specifies their content to be inserted, Ephox recommends the use of a central provider such as [embed.ly](#). While not free, such services provide a enhanced embedded media experience for your users.

## Configuration

Configuration of media content is provided via the [<multimedia>](#) element within an EditLive! configuration file. A separate [<service>](#) configuration file element must be provided for each oEmbed service to be used with EditLive!. Types are evaluated in order, to allow for prioritisation of ambiguous schemes and fallback services to be specified as a last resort.

### Example

The following example would ask youtube.com for how to insert YouTube videos, and embed.ly for all other links.

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <services>
        ...
        <service name="YouTube" endpoint="http://www.youtube.com/oembed" scheme="http://*.
youtube.com/*" />
        <service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed?key=<your key
here>" scheme="*" />
        ...
      </services>
    </multimedia>
    ...
  </mediaSettings>
```

```
...  
</editLive>
```

## See Also

- [<multimedia>](#) Configuration File Element
- [<service>](#) Configuration File Element
- [Instantiating the Applet](#)

# Using Social Media and External Media Services

EditLive! supports embedding of media content from social media services and other web-based media services via the Insert Media dialog.

The Insert Media dialog supports adding content from these services in two ways. For services that support the [oEmbed](#) specification EditLive! can be configured to automatically fetch the required HTML from those services. For other services, the Insert Media dialog supports the insertion of an arbitrary HTML fragment to represent rich media.

For media to be supported via the Media Services portion of the Insert Media dialog the oEmbed service associated with that service must be configured for use with EditLive!.

While EditLive!'s default configuration is pre-configured to support several popular media services including YouTube, Vimeo, Flickr and SlideShare, it's recommended that a commercial media aggregator service like [embed.ly](#) be used in conjunction with this functionality to provide a large breadth of services support. For more information see the article on [Using a Media Aggregator Service](#).

EditLive! supports all four oEmbed types in the specification; photo, video, link and rich.

Note



- The oEmbed "link" type is inserted as a plain hyperlink with no additional information. Editing these hyperlinks is done through the standard hyperlink dialog.
- The oEmbed "photo" type is inserted as an image with no additional information. These images are editable with the built in image editor and their properties can be adjusted using the Image Properties dialog.

Endpoints and URL scheme matching information must be provided for each service provider. This documentation is usually available from the service provider directly.

Media objects embedded within EditLive! are presented to the user in uneditable sections to preserve the output format returned by the service provider. Each section uses an overlay to indicate the type of oEmbed media that has been inserted. The icons can be seen below:

Used to represent rich, interactive and social media. Includes services such as SlideShare, Google Maps and Twitter.

Used to represent video media. Includes services such as YouTube, Vimeo and Brightcove

## Configuration

### Menu and Toolbar Item

To provide the Insert Media menu item or toolbar button, the value *InsertMedia* must be specified for the **name** attribute of either a `<menuItem>` or `<toolbarButton>` configuration file element. For more information on EditLive! for Java configuration files, see the [Instantiating the Applet](#) article. For more information on creating menu or toolbar items, see the [Setting Menu and Toolbar Items](#) and [Menu and Toolbar Item List](#) articles.

### Services Configuration

Configuration of media content is provided via the `<multimedia>` element within an EditLive! configuration file. A separate `<service>` configuration file element must be provided for each oEmbed service to be used with EditLive!. Types are **evaluated in order**, to allow for prioritisation of ambiguous schemes and fallback services to be specified as a last resort. The first match that is encountered will be used to fetch the media asset.

The **scheme** attribute is used to specify the format of the URLs that match to the service and its oEmbed endpoint. For more information about oEmbed schemes and endpoints see the [oEmbed](#) specification.

The sample configuration that ships with EditLive! - *sample\_eljconfig.xml* - includes configuration to use over 20 different services. You can also use a 3rd party oEmbed service like [embed.ly](#) to provide further oEmbed functionality. For more information on using an external oEmbed aggregator see the [Using a Media Aggregator Service](#) documentation.

#### Example

The following example would ask youtube.com for how to insert YouTube videos, and [embed.ly](#) for all other links.

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <services>
        ...
        <service name="YouTube" endpoint="http://www.youtube.com/oembed" scheme="http://*.youtube.com/" />
        <service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed?key=<your key here>" scheme="*" />
        ...
      </services>
    </multimedia>
  </mediaSettings>
</editLive>
```

```
        </multimedia>
    ...
    </mediaSettings>
    ...
</editLive>
```

# Using a Media Aggregator Service

Media aggregator services like [embed.ly](#) provide oEmbed support for a broad range of media services. embed.ly is a commercial media aggregator service that provides oEmbed support for over 250 different services. A [complete listing of supported service providers](#) can be found on embed.ly's site.

It's recommended that media aggregators be provided as a "fallback" service, used to attempt to match URLs for assets that don't match with any other services. A fallback can be specified by using a scheme using the \* wildcard to match all URLs.

## Note

As a scheme of "\*" will match **all** URLs it should be specified at the end of the list of services as services are tried in order until a match is found and "\*" will match all URLs. Also, it is only valid to have a single endpoint associated with the "\*" scheme.

## Example

The following example would ask youtube.com for how to insert YouTube videos, and embed.ly for all other links.

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <services>
        ...
        <service name="YouTube" endpoint="http://www.youtube.com/oembed" scheme="http://*.youtube.com/" />
        <service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed?key=<your key here>" scheme="*" />
        ...
      </services>
    </multimedia>
    ...
  </mediaSettings>
  ...
</editLive>
```

## Configuring for Embed.ly Usage

Embed.ly is a commercial service and a key to access this service can be obtained directly from [embed.ly](#). This key needs to be specified in as part of the endpoint URL for the Embed.ly service:

```
<service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed?key=<your key here>" scheme="*" />
```

# Embedding Media Using HTML5

The Insert Media dialog supports embedding media via the HTML5 <video> and <audio> tags. This is achieved through the Video and Audio tabs on the dialog respectively.

Media content that's inserted into EditLive! must be first uploaded to a web server. Files local to the user's machine cannot be used as either video or audio assets in the content.

For each tag, the dialog supports the use of multiple source files of different formats and HTML to provide subtitles or fallback support in the case of browsers that do not support HTML5 media.

## Inserting Media

Once media has been inserted via the dialog, it will render with a place holder image in EditLive!.


Video content renders as an area with the specified height and width and with the following placeholder:



Audio content renders as a placeholder set of audio controls:



### Media Playback

 Media can only be played in the Preview tab of EditLive!

EditLive! supports the insertion of the following formats.

The following video formats are supported:

- MP4
- OGG
- WebM

The following audio formats are supported:

- WAV
- MP3
- OGG

# Advanced Media Support

EditLive! includes support for a number of advanced media insertion techniques. These include the ability to embed complex object types, link to content within WebDAV enabled content repositories or create your own image browsing component.

Information on these techniques can be found here:

- [Object Tag Support](#)
- [Using WebDAV with EditLive!](#)
- [Enabling WebDAV on a Web Server](#)
- [Image Insertion Dialog's Browser Component](#)



# Object Tag Support

EditLive! supports the embedding of multimedia content using <object> tags. This section provides information on how the object tag support of EditLive! functions and how to configure object tags within EditLive!.

## Object Tag Overview

EditLive! supports the embedding of any form of multimedia content via the <object> tag and associated <param> tags, provided that the relevant configuration information has been provided for the format. Multimedia objects can be inserted from local files that will be uploaded when the content from EditLive! is submitted using a [HTTP Post](#). For more information on specifying how multimedia is uploaded to a server, see the [HTTP Upload Support for Images and Objects](#) article.

Object tags embedded within EditLive! are presented to the user as an icon within a resizable area. The icon can be seen below:



While most of the parameters and properties for the object type are configured via the EditLive! configuration file, the **height**, **width** and **data** attributes of the <object> tag must be specified by the user through the object insertion dialog. The **height** and **width** are used to provided the dimensions, if any, for the object, while the **data** attribute contains the URL for the location of the object source file.

To create an Insert Object menu item or toolbar button, the value *InsertObject* must be specified for the **name** attribute of either a <menuitem> or <toolbarButton> configuration file element. For more information on EditLive! for Java configuration files, see the [Instantiating the Applet](#) article. For more information on creating menu or toolbar items, see the [Setting Menu and Toolbar Items](#) and [Menu and Toolbar Item List](#) articles.

## Configuration

Configuration information of object tags is provided via the <multimedia> element within an EditLive! configuration file. A separate <type> configuration file element must be provided for each object type to be used with EditLive!. Types are differentiated by their unique **name** attribute.

### Example

The following example would allow users to insert AVI or WAV objects through the Insert Object dialog.

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <types>
        <type
          name="AVI"
          type="application/x-mplayer2"
          extension="avi"
          allowCustomParams="true" />
        <type
          name="WAV Audio"
          type="application/x-mplayer2"
          extension="wav"
          allowCustomParams="true" />
      </types>
    </multimedia>
    ...
  </mediaSettings>
  ...
</editLive>
```



<object> tags created in HTML can also have nested <param> tags to specify additional information. For example:

```
<html>
  <body>
    ...
    <object type="video/quicktime">
      <param name="volume" value="10" />
    </object>
    ...
  </body>
</html>
```

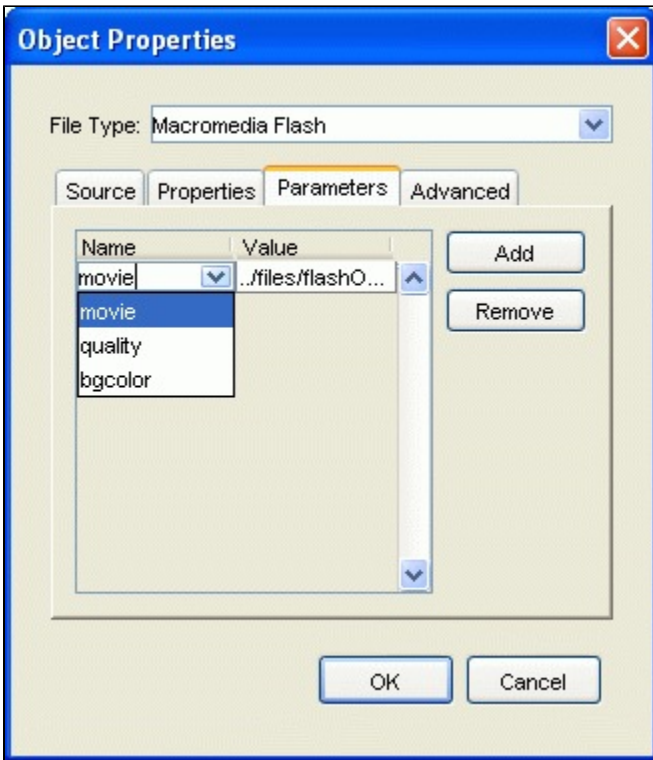
To allow users to specify <param> tags in the Insert Object dialog, each <type> element can have any number of nested <param> elements in the configuration file. The **name** attribute of this configuration file element will map to the **name** attribute of the generated HTML <param> tag. Through the Insert Object dialog the user can specify the value that will map to the **value** attribute of the generated HTML <param> tag.

#### Example

The following example shows how to allow users to specify an optional 3 <param> tags for an Adobe Flash <object> tag.

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <types>
        ...
        <type
          name="Adobe Flash"
          type="application/x-shockwave-flash"
          extension="swf"
          allowCustomParams="true"
          urlParam="movie" >
          <param name="movie" />
          <param name="quality" />
          <param name="bgcolor" />
        </type>
        ...
```

```
</types>
</multimedia>
</mediaSettings>
...
</editLive>
```



Please see the [<type>](#) and [<param>](#) configuration file elements for more examples of specifying object tags to be used with the Insert Object dialog of EditLive!.

## Object File Sources

Objects can be inserted either from the file system of the client machine.

In order for local files to be inserted as multimedia objects in EditLive! the file upload properties must be correctly configured. For more information on this, please see the article on [HTTP Upload Support for Images and Objects](#).

### See Also

- [<multimedia>](#) Configuration File Element
- [<type>](#) Configuration File Element
- [<param>](#) Configuration File Element
- [Instantiating the Applet](#)
- [HTTP Upload Support for Images and Objects](#)

# Using WebDAV with EditLive!

WebDAV support has been removed in EditLive! 9.1

Tiny EditLive! supports the WebDAV protocol. The WebDAV protocol allows easy directory browsing of server locations. WebDAV can be used in EditLive! to enable directory browsing from a server location when adding images or hyperlinks to a document. This provides the end users of EditLive! with an interface to easily browse directories on the server. EditLive! can also be configured to allow all local images in the user's content to be uploaded to a WebDAV repository.

Through the use of EditLive!'s configuration files users can also be restricted in their access so that only specific WebDAV repositories are available to them.

This document provides information on how to use WebDAV with EditLive!. It assumes that you have a WebDAV enabled server and are able to configure your server to allow WebDAV access to specific directories.

## Using WebDAV with Images in EditLive!

Using a WebDAV server with an instance of EditLive! that has been configured accordingly results in users being able to browse the relevant WebDAV repository from within the *Insert Object*, *Insert Image* and *Insert Hyperlink* dialogs in EditLive!. In the case of the *Insert Image* and *Insert Object* dialogs, EditLive! filters the available files on the WebDAV repository according to their MIME type. The *Insert Image* dialog will only include files which have the *image/jpeg*, *image/gif* or *image/png* MIME types while the *Insert Object* dialog will only include files that match the file types specified in the `<multimedia>` configuration file element and its children.

## Configuring EditLive! for Use with WebDAV Browsing

EditLive! can be easily configured for use with WebDAV via the EditLive! configuration file. The configuration settings for the use of WebDAV with EditLive! can be found within the `<webdav>` configuration file element. The `<webdav>` element contains a listing of WebDAV repositories which have their details specified by the `<repository>` configuration file elements.

### Basic Configuration Example

The following provides a basic example of how to configure an instance of EditLive! for use with a WebDAV repository. It involves the minimum number of settings to get WebDAV functioning correctly within EditLive!. In this example the server does not implement password protection. For the purposes of this example the WebDAV server which EditLive! is being configured for use with has the following properties:

- The WebDAV repository has the URL `http://www.yourserver.com/UserFiles/WebDAV`.
- The base for documents created with EditLive! (i.e. the calling of the `setBaseURL Method`) is `http://www.yourserver.com/UserFiles/EditLiveFiles`. This means that the location of the WebDAV repository relative to the EditLive! document base is `../WebDAV`.
- The repository should be listed to users as the *Images* repository.

The EditLive! configuration file, in this case, would contain the following elements:

```
<editLive>
...
<webdav>
  <repository
    name="Images"
    baseDir="http://www.yourserver.com/UserFiles/WebDAV"
    webDAVBaseURL=" ../WebDAV" />
  </webdav>
...
</editLive>
```

### Setting a Default Browsing Directory

If you want the EditLive! end users to view a directory other than the root directory of the WebDAV repository by default, then the `defaultDir` attribute of the `<repository>` configuration file element should be used and assigned the relevant value. Users can still move up the directory tree to the root directory if desired.

Continuing from the example above, if the `http://www.yourserver.com/UserFiles/WebDAV` directory had a subdirectory *images* which you wished the usersto access by default then the XML configuration would be as follows:

```
<editLive>
...
<mediaSettings>
  <images>
    <webdav>
      <repository
        name="Images"
```

```

        baseDir="http://www.yourserver.com/UserFiles/WebDAV"
        webDAVBaseURL=" ../WebDAV"
        defaultDir="images" />
    </webdav>
</images>
</mediaSettings>
...
</editLive>

```

## MIME Type Filtering with WebDAV

The browsing of a WebDAV repository with EditLive! can be restricted according to the MIME of the files within the repository. As the WebDAV functionality within EditLive! is used with images then files with the following MIME types will be displayed:

- *image/jpeg*
- *image/png*
- *image/gif*

In order to activate MIME type filtering within EditLive! the configuration file must contain the relevant setting. Continuing from the examples above the XML configuration for EditLive! would be as follows:

```

<editLive>
...
<mediaSettings>
  <images>
    <webdav>
      <repository
        name="Images"
        baseDir="http://www.yourserver.com/UserFiles/WebDAV"
        webDAVBaseURL=" ../WebDAV"
        defaultDir="images"
        useMimeType="true" />
    </webdav>
  </images>
</mediaSettings>
...
</editLive>

```

The default setting for the **useMimeType** attribute is *true*.

## Password Protected WebDAV Repositories

If your WebDAV repository implements basic authentication then you can configure EditLive! to use the correct username and password information. These values are specified using the **username** and **password** attributes, respectively, of the **<repository>** configuration file element. If the username and password specified are incorrect, EditLive! will prompt the user for a username and password when the WebDAV server is accessed.

EditLive! for Java supports the following forms of authentication:

- Basic
- Digest
- NTLM

Continuing from the examples above, if the realm was *www.yourserver.com* (an NTLM realm), and if the username was *webdav* and the corresponding password was *example*, then the XML configuration for EditLive! would be as follows:

```

<editLive>
...
<mediaSettings>
  <images>
    <webdav>
      <repository
        name="Images"
        baseDir="http://www.yourserver.com/UserFiles/WebDAV"
        webDAVBaseURL=" ../WebDAV" defaultDir="images" useMimeType="true"
        username="webdav" password="example" />
    </webdav>
  </images>
</mediaSettings>
...
</editLive>

```

```
    </webdav>
  </images>
</mediaSettings>
...
</editLive>
```

## Configuring EditLive! for Uploading Images to WebDAV Repositories

EditLive! can be easily customized to upload any local images to a specified WebDAV repository. Using an EditLive! configuration file, you can nest a [<repository>](#) element within the [<httpUpload>](#) element. Note that unlike other uses of WebDAV in EditLive!, there is no browser. WebDAV image uploads can only be done to a single folder.

### See Also

- [<webdav>](#) Configuration File Element
- [<repository>](#) Configuration File Element
- [<httpUpload>](#) Configuration File Element
- [setBaseUrl Method](#)

# Enabling WebDAV on a Web Server

WebDAV support has been removed in EditLive! 9.1

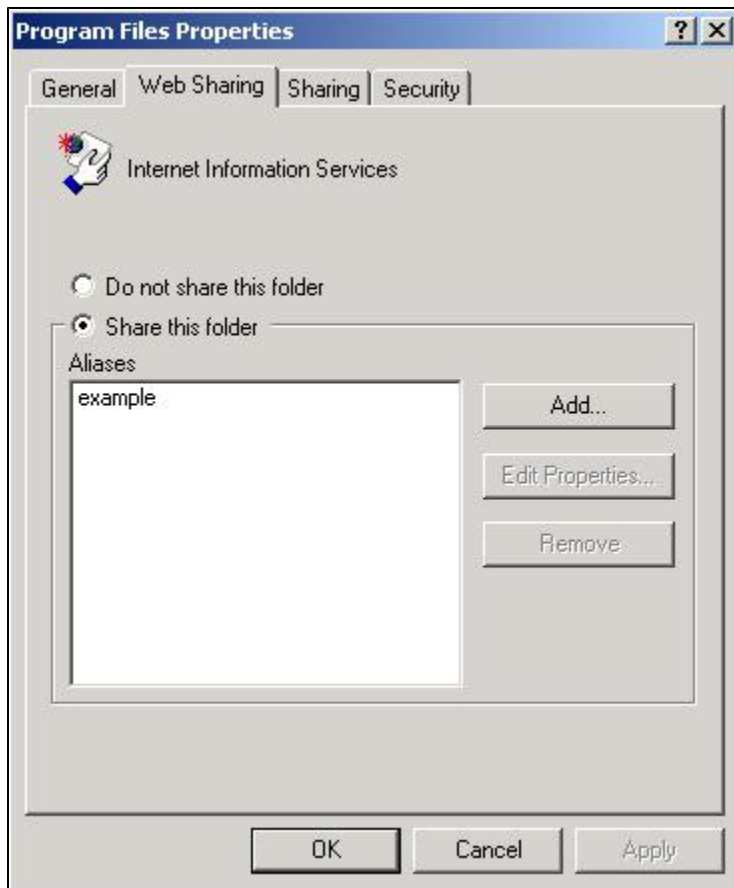
WebDAV is a set of extensions to the HTTP protocol which allows for the collaborative editing of files and management of files on remote Web servers. For more information please refer to <http://www.webdav.org>.

This document provides information on how WebDAV can be enabled for use on a selection of Web servers and is intended as a basic guide only. If your Web server is not listed within this documentation, or if you wish to obtain more information about your Web server's WebDAV support, please consult the documentation for your Web server. If the steps outlined in this document are not correct for your Web server, please consult your Web server's documentation for information on its WebDAV support.

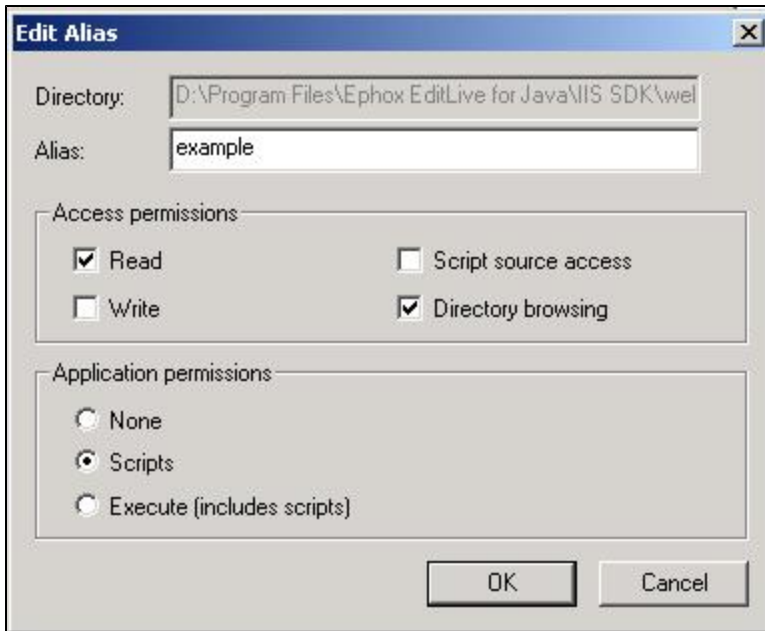
## Microsoft IIS 5.0

Microsoft Internet Information Services (IIS) version 5.0 includes support for WebDAV. IIS has WebDAV enabled on the server by default. To enable the WebDAV functionality of IIS 5.0 for a specific directory, the relevant directory must be available through your Web server and must also have directory browsing turned on.

A folder may be shared on a Web server by editing its properties. The relevant properties appear on the Web Sharing tab. In the example below, a folder has been shared on an IIS Web server using the Web alias of *example*.



The example below demonstrates the enabling of WebDAV for the folder with the alias of *example*.



The configuration of WebDAV for a directory on IIS is achieved through the Web server permission settings.

## Apache Tomcat

Apache Tomcat includes WebDAV functionality in versions 4.0 and above. The default install of Apache Tomcat includes an example WebDAV application in the `TOMCAT_HOME/webapps/webdav` directory, where `TOMCAT_HOME` is the install directory of Apache Tomcat.

The WebDAV application packaged with Apache Tomcat is configured to provide read-only access. To configure this example to allow for write access, uncomment the following lines in the `TOMCAT_HOME/webapps/webdav/WEB_INF/web.xml` file:

```
...
<!--
<init-param>
  <param-name>readonly</param-name>
  <param-value>>false</param-value>
</init-param>
-->
...
```

Please consult the Apache Tomcat documentation for more information on configuring Apache Tomcat.

## Apache Web Server

A third-party Apache module called `mod_dav` is available to enable DAV functionality with an Apache Web server. The distribution for this module can be found at [http://www.webdav.org/mod\\_dav/](http://www.webdav.org/mod_dav/). For information on how to install the `mod_dav` module please see its [install page](#).

To turn WebDAV functionality on for a directory once you have installed the `mod_dav` module, simply place the following code within the relevant `<Directory>` or `<Location>` directive in your Apache configuration file (`httpd.conf`):

```
DAV On
```

Once WebDAV is enabled for a `<Directory>` then all its subdirectories will also have WebDAV enabled. When enabling WebDAV for a `<Location>`, WebDAV will be enabled for that portion of the URL namespace.

For more information on Apache Web Server WebDAV configuration see [http://www.webdav.org/mod\\_dav/install.html#apache](http://www.webdav.org/mod_dav/install.html#apache).

See Also

- [Using WebDAV with EditLive!](#)
- <http://www.webdav.org>
- [http://www.webdav.org/mod\\_dav/](http://www.webdav.org/mod_dav/)
- [http://www.webdav.org/mod\\_dav/install.html](http://www.webdav.org/mod_dav/install.html)



# Image Insertion Dialog's Browser Component

EditLive! allows developers to specify a web page that can be viewed through EditLive!'s image insertion dialog. This allows developers to integrate web pages featuring image hyperlinks into the image insertion architecture of EditLive!. To insert images from the image browser, users simply have to click on the image hyperlink and then click the OK button on the image insertion dialog.

## Why Use the Image Browser?

This functionality is useful for developers who already possess access to an online image repository and wish to easily access these images through the EditLive! interface. Developers may also wish to create their own web page in order to make a commonly used collection of images readily available to a user base.

Developers can also specify if the location of images inserted from a web page are relative or absolute. This can be useful in ensuring that all the required information relating to the location of the image is passed back to EditLive!. More information on relative and absolute referencing of images inserted from a web page can be found in the [<imageBrowser>](#) configuration file element article.

## Configuring the Image Insertion Dialog's Browser Component

Detailed information on the image insertion dialog's browser component can be located in the [<imageBrowser>](#) configuration file element article.

## Restrictions on Web Pages Viewed Using the Image Browser

Web pages referenced through the image insertion dialog's browser component face some restrictions in order to operate correctly.

- Any JavaScript contained in these web pages will not function.
- Images can only be inserted from web pages by image hyperlinks.

### Example

```
<a href="images/ephoxImage.gif">
```

- Image tags cannot be clicked to obtain a reference to the image location.

### Example

The following would not return any information to EditLive! upon being clicked.

```

```

See Also

- [<imageBrowser>](#) Configuration Element

# Collaboration

- [Getting Started With Track Changes](#)
- [Track Changes Serialization Format](#)
- [Commenting](#)

# Getting Started With Track Changes

## What is Track Changes?

The Track Changes functionality of EditLive! captures change information as users make changes to a document. This makes it easier for users to create and review documents as part of a collaboration or workflow process. This functionality has been designed to enhance the collaborative functionality of content applications and is easily integrated with any application without changes to the underlying repository and other systems.

Once a document has changes tracked, users can perform operations on those changes. Users can browse the changes in the document and can accept or reject each individual change.

EditLive! tracks changes made by the user when editing in the Design view of the editor. When creating and editing content in the Code view of the editor, these changes are not tracked.

In order for users to utilize the track changes functionality of the editor, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

## How Are Changes Recorded?

EditLive! records changes as one of three types:

- *Inserted content* – Rendered as underlined text
- *Removed content* – Rendered as strikethrough text
- *Formatting changes* – Rendered as highlighted text (some formatting changes, such as changing the indent for HTML list items, apply no additional rendering to indicate a tracked change. These changes are, however, still tracked by the editor).

Each change is rendered within a color representing the user who made the change. Changes also record with the name of the user who made the change and the time the change was made.

The Track Changes functionality of EditLive! provides more detailed change information than the document comparison functionality of other systems. Many comparison engines can only approximate changes, particularly when dealing with the flow of natural language. As EditLive! captures change information at the time users make the change, it can provide more accurate and detailed change information.

## How Do I Turn on Track Changes?

There are a number interface commands associated with Track Changes which can be added to the menus and toolbars of EditLive! to provide users with access to the functionality. A list of these is available below. These commands enable users to turn change recording on and off, accept and reject changes, and browse the changes in the document.

Users can enable or disable track changes at any time via the [Enable Track Changes](#) interface item. When a HTML document is loaded into EditLive!, Track Changes is turned off by default. You can override this behavior and turn Track Changes on by default through the use of the `enableTrackChanges` attribute of the `<wysiwygEditor>` configuration file element. You can also prevent users from turning Track Changes functionality on/off by removing the [Enable Track Changes](#) command from the interface.

If the document contains track changes information from a previous EditLive! editing session then track changes will be automatically turned on.

See the [Menu and Toolbar Item List](#) article for a complete list of the available menu items and toolbar buttons available for EditLive! related to Track Changes.

## What Do I Need to Do to Work With Track Changes In My Application?

The Track Changes functionality of EditLive! has been designed so that it does not require any changes to the architecture of the underlying content application. All change information is stored within an element at the end of the HTML document's `<body>`. For more information on the track changes mark up format see the document on the [Track Changes Serialization Format](#).

If you are integrating the EditLive! Track Changes functionality with a system which makes use of usernames then you can pass through a username to EditLive! which will then be associated with any changes made by that user. To achieve this, set the [setUserName Method](#) of EditLive! at load time. Note that setting this username will override any username which has been previously specified by the user. Also, to prevent the user from changing their username, the **Set Username...** command should be removed from the EditLive! interface. For more information on customizing the menu item and toolbar commands for an instance of EditLive! see the [Menu and Toolbar Item List](#).

If two users enter the same name (either through the `UserName` load-time property or using the **SetUsername...** toolbar/menu item), the changes made by these two separate users will render as if the changes were made by the same user. In order for each user to have their changes uniquely tracked and rendered, each user will need to enter a unique username.

# Track Changes Serialization Format

In order to perform the track changes functionality for EditLive!, Tiny has generated a simple serialization format for HTML documents to encapsulate change information. This serialization format is based on the same principals as the OpenOffice.org serialization.

Change information is stored within the HTML document by using <change> tags. The end of the <body> element of the HTML document contains a hidden <div> tag containing the logic for the change information. Change tags are mapped by their id attributes to corresponding tags with the same id attribute value within the hidden <div>. This allows EditLive! to easily merge the user's changes by nesting <change> tags while mapping the change details in the hidden <div>.

## Example HTML Document

The following is an example HTML document containing change information created using EditLive!.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <base href="http://testsuite/java" />
    <meta http-equiv="Content-Type" content="text/html; charset=cp1252" />
    <meta name="generator" content="EditLive! 6.0.0.1186" />
  </head>
  <body>
    <p>
      Line 1 - <strong>BOLD</strong> text<change id="352" type="start" />
    </p>
    <p>
      new line of text added<change id="352" type="end" />
    </p>
    <p>
      <change id="379" type="both" />
    </p>
    <p>
      Line 3 - <span style=" text-decoration: underline;">UNDERLINED</span> text
    </p>
    <p>
      Line 4 - Hyperlink <change id="381" type="start" /><a href="http://www.ephox.com/support">here<
/a><change id="381" type="end" />
    </p>

    <div style="display: none;"><trackchanges version="1.0"><insert id="352" xml:space="preserve" author="Ephox"
time="2006-09-19T14:04:09+1000"/><remove id="379" xml:space="preserve" author="Ephox" time="2006-09-19T14:04:
17+1000" type="range"><p>Line 2 - <em>ITALIC</em> text</p></remove><multiplepropertychange id="380" xml:space="
preserve" author="Ephox" time="2006-09-19T14:04:25+1000"><setproperties id="381" xml:space="preserve" author="
Ephox" time="2006-09-19T14:04:25+1000"><propertychange original="http://www.ephox.com" tag="a" new="http://www.
ephox.com/support" name="href" /></setproperties></multiplepropertychange></trackchanges></div>
  </body>
</html>
```

## Using HTML Documents in EditLive! Featuring Change Information

### Extracting HTML from EditLive! Featuring Change Information

Change information is encapsulated in the XHTML in a way that will not cause any difference in rendering when the XHTML is viewed through a web browser. This allows users to publish their content to the web with change information still intact, and later edit this content and still view all changes made. All change information is stored within the <BODY> element of the document. Hence, when using the [getBody Method](#), all change information will still be extracted from the editor.

### Inserting HTML Featuring Change Information into EditLive!

When HTML is loaded into EditLive! (using any of the methods listed in the [Setting EditLive! Contents](#) article), if change information is identified Track Changes will automatically be enabled and Track Changes rendering will be initiated.

## Change Types

There are three different changes that can occur to the content in EditLive!:

- *Changes that Add Content* - These changes include actions such as typing new text or adding a new element (such as a table or a list).
- *Changes that Remove Content* - These changes include deleting text or removing elements.

- *Changes that Adjust Properties* - Property adjustment changes that refer to changing the attributes of a HTML tag (e.g. changing the src attribute for an <img> tag)

This article will show how each of these change types is stored in the XHTML document.

## Changes that Add Content

### Example

The user loads the following HTML fragment into EditLive!:

```
<p>
    Line 1 - <strong>BOLD</strong> text
</p>
```

The user then places the cursor after the string text. The user hits enter and types new line of text added. The HTML document would now appear as follows:

```
<p>
    Line 1 - <strong>BOLD</strong> text<change id="313" type="start" />
</p>
<p>
    new line of text added<change id="313" type="end" />
</p>
<div style="display: none;"><trackchanges version="1.0"><insert id="313" xml:space="preserve" author="Ephox"
time="2006-09-19T13:20:18+1000" /></trackchanges></div>
```

## Changes that Remove Content

### Example

The user loads the following HTML fragment into EditLive!:

```
<p>
    Line 1 - <strong>BOLD</strong> text
</p>
```

The user then selects the line of text and presses the backspace key. The HTML document would now appear as follows:

```
<p>
    <change id="336" type="both" />
</p>
<div style="display: none;"><trackchanges version="1.0"><remove id="336" xml:space="preserve" author="Ephox"
time="2006-09-19T13:39:20+1000" type="range"><removedContent><p>Line 1 - <strong>BOLD</strong> text</p><
/removedContent></remove></trackchanges></div>
```

### Example

The user loads the following HTML fragment into EditLive!:

```
<p>paragraph 1</p>
<ul>
    <li>list item 1</li>
    <li>list item 2</li>
</ul>
<p>paragraph 2</p>
```

In the editor's design view, the user places the cursor in front of the text paragraph 2 and presses the delete key. The HTML document would now appear as follows:

```
<p>paragraph 1</p>
```

```

<ul>
<li>list item 1</li>

<li>list item 2<change type="both" id="0" />paragraph 2</li>
</ul>

<div style="display: none;"><trackchanges version="1.0"><remove id="0" xml:space="preserve" author="Ephox"
time="2006-10-26T09:42:59+1000" type="range"><removedContent><p>
</p></removedContent><closeTags><ul><li/></ul></closeTags><openTags><p/></openTags></remove></trackchanges><
/div>

```

## Changes that Adjust Properties

### Example

The user loads the following HTML fragment into EditLive!:

```

<p>
  Line 1 - Hyperlink <a href="http://www.ephox.com">here</a>
</p>

```

The user then selects the hyperlink and opens the Tiny hyperlink dialog. The user then changes the Address field to contain `http://www.ephox.com/support`. After clicking OK, the HTML document would now appear as follows:

```

<p>
  Line 1 - Hyperlink <change id="349" type="start" /><a href="http://www.ephox.com/support">here<
/a><change id="349" type="end" />
</p>
<div style="display: none;"><trackchanges version="1.0"><multiplepropertychange id="348" xml:space="preserve"
author="Ephox" time="2006-09-19T13:45:11+1000"><setproperties id="349" xml:space="preserve" author="Ephox"
time="2006-09-19T13:45:11+1000"><propertychange original="http://www.ephox.com" tag="a" new="http://www.ephox.
com/support" name="href" /></setproperties></multiplepropertychange></trackchanges></div>

```

### Example

The user loads the following complete HTML Document into EditLive!:

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="stylesheet" href="main.css" type="text/css" />
<meta content="EditLive! for Java 6.0.0.1423" name="generator" />
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />
<style type="text/css">
<!--
h5.warning {
font-weight: bold;
color: red;
}
-->
</style>
</head>
<body>
<h5>Heading 5 - Text</h5>
</body>
</html>

```

The user then switches to design view, selects the text *Heading 5 - Text* and changes the style to `h5.warning`. The HTML document would now appear as follows:

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="stylesheet" href="main.css" type="text/css" />
<meta content="EditLive! 6.0.0.1423" name="generator" />
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

```

```
<style type="text/css">
<!--
h5.warning {
  font-weight: bold;
  color: red;
}
-->
</style>
</head>
<body>
<h5 class="warning" changeid="1">Heading 5 - Text</h5>

<div style="display: none;"><trackchanges version="1.0"><setproperties id="1" xml:space="preserve" author="
Ephox" time="2006-10-26T10:39:30+1000"><propertychange new="warning" name="class"/></setproperties><
/trackchanges></div>
</body>
</html>
```

# Commenting

The Commenting feature of EditLive! allows users to add comments to selections of text in a document. This feature integrates with the [Track Changes](#) feature by using the same usernames and colors to identify comments within the document.

In order for users to utilize the Commenting functionality, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period, or
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

## Enabling Commenting

There are two steps to perform in order to enable Commenting.

### Enabling the Commenting Plugin

Commenting is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *imageEditor* for a `<plugin>` element resident in your EditLive! [Configuration File](#).

Example

```
<editlive>
  ...
  <plugins>
    <plugin name="commenting" />
  </plugins>
</editlive>
```

### Specifying the Commenting Menu Items

The [Menu and Toolbar Item List](#) features two menu items for Commenting. By default, both menu items will appear in the Track Changes menu. In addition, the **Add Comment...** menu item will also appear in the [shortcut \(context\) menu](#). For adding comments, we recommend using the shortcut menu as it allows the user to select text, right-click on it, and open the Commenting pop-up. The Commenting pop-up will appear next to the selected area of text regardless of which menu this menu item is placed on.



# Web Content Accessibility

- [Accessibility Compliance](#)
- [Accessibility As You Type](#)
- [Table Accessibility](#)

# Accessibility Compliance

In order for users to utilize the Accessibility functionality, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period, or
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

EditLive! allows users to check their document against the [W3C Accessibility Compliance Guidelines 2.0](#). The functionality aides users in creating documents which comply with the guidelines stipulated by the W3C and the US Government ITAW. See the [Menu and Toolbar Item List](#) article for more information on how to enable accessibility compliance checking through EditLive!.

Users should be aware that the information provided through EditLive!'s accessibility compliance checking is intended to be a guide only. Tiny has been implemented with reference to [W3C Accessibility Compliance Guidelines](#) and [United States Section 508 accessibility guidelines](#); however, elements of this implementation may rely on interpretation. The user's interpretation of these guidelines may differ from the Tiny interpretation. Users are encouraged to follow the links provided with each accessibility issue EditLive! reports for extended information on the guidelines. These links will lead them to either the specific W3C Web Content Accessibility Guidelines checkpoint on the W3C website or the appropriate United States Section 508 checkpoint.

Important!



Web Content Accessibility Guidelines recommend that the proper HTML elements should be used to mark up emphasis: `<EM>` and `<STRONG>`. The `<B>` and `</>` elements should *not* be used; they are used to create a visual presentation effect. The `<EM>` and `<STRONG>` elements were designed to indicate structural emphasis that may be rendered in a variety of ways (font style changes, speech inflection changes, etc.) - [H49: Using semantic markup to mark emphasized or special text | Techniques for WCAG 2.0](#).

To ensure that EditLive! uses logical emphasis for the application of bold and italics, please set the `logicalEmphasis` attribute of the `<htmlFilter>` configuration file element to `true`.

# Accessibility As You Type

Accessibility As You Type allows elements in the editor's document to display icons representing their accessible state. Currently Accessibility As You Type supports feedback for the current HTML elements:

- <IMG>
- <TABLE> (and its child elements)
- <DIV>

In order for users to utilize the Accessibility As You Type functionality, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period, or
- An [Enterprise Edition](#) license has been installed for the editor. See the [Licensing EditLive!](#) article for more information.

## Enabling Accessibility As You Type

There are two steps to perform in order to enable Accessibility As You Type.

### Enabling the Accessibility As You Type Plugin

Accessibility As You Type is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *accessibility* for a [<plugin>](#) element resident in your EditLive! [Configuration File](#).

```
<editlive>
  ...
  <plugins>
    <plugin name="accessibility" />
  </plugins>
</editlive>
```

### Specifying the Accessibility As You Type Toolbar and/or Menu Item

The [Menu and Toolbar Item List](#) features a toggle toolbar/menu item for activating or deactivating Accessibility As You Type.

## Modifying Accessibility As You Type

### Enabling/Disabling Accessibility As You Type by Default

The [<accessibilityChecks>](#) configuration file element provides developers with an option for enabling/disabling Accessibility As You Type by default.

### Adjusting the Error/Warning Level for Accessible Elements

The [<accessibilityChecks>](#) configuration file element also provides developers with options for whether specific accessibility checks will return errors, warnings, or nothing.

# Table Accessibility

Table Accessibility features an assortment of tools for ensuring that tables within your HTML document conform to various accessibility guidelines (such as the [W3C Accessibility Compliance Guidelines](#) and the [United States Section 508 Accessibility Guidelines](#)).

Table Accessibility features the following **Enterprise Edition only** components:

- [Apply Cell Headers](#)
- [Display Header to Data Mappings](#)

All remaining Table Accessibility functionality outlined in the [Menu and Toolbar Item List](#) is available in the standard install of EditLive!.

In order for users to utilize the Apply Cell Headers and Display Header to Data Mappings functionalities, one of the following conditions must be met:



- Users are still operating the editor in their 30 day trial period, or
- An [Enterprise Edition](#) license has been installed for the editor. See the [Licensing EditLive!](#) article for more information.

## Enabling Table Accessibility

Two steps must be performed in order to enable Table Accessibility.

### Enabling the Table Accessibility Plugin

Table Accessibility is a [plugin](#) packaged with EditLive!. Because this plugin is packaged by default, you can simply specify the **name** attribute *tableToolbar* for a `<plugin>` element resident in your EditLive! [Configuration File](#).

```
<editlive>
  ...
  <plugins>
    <plugin name="tableToolbar" />
  </plugins>
</editlive>
```

### Specifying the Table Accessibility Toolbar and/or Menu Items

The [Menu and Toolbar Item List](#) features a toggle toolbar/menu item for activating or deactivating Table Accessibility.

# Internationalization Support

- [Internationalization Support Overview](#)
- [Character Sets Supported](#)
- [Specifying Character Sets for Internationalization](#)
  - [Specifying Character Sets in the Applet](#)
  - [Specifying Character Sets in the Swing SDK](#)

# Internationalization Support Overview

Tiny EditLive! includes various features allowing for increased ease of use of EditLive! by international users. The internationalization features of EditLive! include user interface translations in several languages, international spell checkers, and support for international character sets.

## Interface Translations

The interface for EditLive! has been translated into several different languages as part of the program's internationalization support. The language of the editor is dependant on the language setting of the client system's operating system. If the language setting of the client system specifies a language which is not supported by EditLive!, the interface will default to English.

To ensure menu items displayed in the editor are internationalized, you will need to use pre-defined menu names in your configuration file. For more information, see the [<menu> Configuration File Element](#).

The interface for EditLive! is internationalized for the following languages:

- Arabic
- Catalan
- Chinese (Simplified)
- Chinese (Traditional)
- Croatian
- Czech
- Danish
- Dutch
- English
- Farsi
- Finnish
- French (Europe)
- German
- Greek
- Hebrew
- Hungarian
- Italian
- Japanese
- Kazakh
- Korean
- Norwegian
- Polish
- Portuguese (Brazil)
- Portuguese (Europe)
- Romanian
- Russian
- Slovak
- Slovenian
- Spanish (Europe)
- Swedish
- Turkish
- Thai
- Ukrainian

## Supported Character Sets

EditLive! provides a variety of character sets for representing HTML content. See the [Character Sets Supported](#) article for a complete list of the character sets supported by EditLive!. For detailed information on how to specify the character set used by EditLive!, see the [Specifying Character Sets in the Applet](#) article.

## Supplied Spell Checking Dictionaries

EditLive! comes packaged with several language-specific dictionaries. These dictionaries are used in conjunction with EditLive's spell checking functionality. See the [Tiny Dictionaries](#) article for a list of the dictionaries available for use with EditLive!. For information on how to specify which dictionary is used with EditLive!, see the [<spellCheck \(Applet\)> Configuration File Element](#).

## Supplied Thesauruses

EditLive! for Java comes packaged with several language specific-thesauruses. See the [Tiny Thesauruses](#) article for a list of the thesauruses available for use with EditLive!. For information on how to specify which thesaurus is used with EditLive!, see the [<thesaurus \(Applet\)> Configuration File Element](#).

See Also

- [Character Sets Supported](#)
- [Specifying Character Sets in the Applet](#)
- [Tiny Dictionaries](#)
- [<menu> Configuration File Element](#)
- [<spellCheck \(Applet\)> Configuration File Element](#)
- [<thesaurus \(Applet\)> Configuration File Element](#)



# Character Sets Supported

## EditLive! Supported Character Sets

EditLive! supports the display and usage of the following character sets:

Character Set	Character Set Name
ASCII	American Standard Code for Information Interchange
CP1252	Windows Latin-1
UTF-8	Eight-bit Unicode Transformation Format
UTF-16	Sixteen-bit Unicode Transformation Format
ISO2022CN	Sixteen-bit Unicode Transformation Format
ISO2022JP	JIS X 0201, 0208 in ISO 2022 form, Japanese
ISO2022KR	ISO 2022 KR, Korean
ISO8859_1	ISO 8859-1, Latin Alphabet No.1
ISO8859_2	ISO 8859-2, Latin Alphabet No.2
ISO8859_3	ISO 8859-3, Latin Alphabet No.3
ISO8859_4	ISO 8859-4, Latin Alphabet No.4
ISO8859_5	ISO 8859-5, Latin/Cyrillic Alphabet
ISO8859_6	ISO 8859-6, Latin/Arabic Alphabet
ISO8859_7	ISO 8859-7, Latin/Greek Alphabet
ISO8859_8	ISO 8859-8, Latin/Hebrew Alphabet
ISO8859_9	ISO 8859-9, Latin Alphabet No.5
ISO8859_13	ISO 8859-13, Latin Alphabet No.7
ISO8859_15	ISO 8859-15, Latin Alphabet No.9
SJIS	Shift-JIS, Japanese
Big5	Chinese Big5

If a character is supported by EditLive!'s defined character set, yet the current font-face being used cannot render this character, the character will not be corrupted and simply rendered as .

If a character is not supported by EditLive!'s defined character set, the character will be corrupted and replaced with a ?. This corruption is **not** reversible. Please take care to define the correct character set for your content.

## See Also

- [Specifying Character Set](#)
- [setOutputCharset Method](#)



# Specifying Character Sets for Internationalization

# Specifying Character Sets in the Applet

Ephox EditLive! supports multiple character sets which allow it to be used in an international environment. Specifying the character set used by an instance of EditLive! defines which characters the editor can correctly render.

For a comprehensive list of the character sets supported by EditLive!, please see the [Character Sets Supported](#) article.

## Default Character Set

If a character set is not specified, the default character set for the user's JVM will be applied to the editor. This means the editor will exhibit different behaviour depending on the system language of the current user. On English Windows systems, the default character set is ISO8859\_1 (also known as CP1252). If, for example, you attempt to load content urlencoded as UTF-8 into the editor without specifying the UTF-8 characters set, a user loading the editor on English Windows would corrupt all unicode characters - eg "à" will be turned into "Ã¶". This corruption is not reversible.

## Specifying EditLive!'s Character Set

The character set used by EditLive! can be specified by the following:

- Using the [setDocument Method](#)  
A <meta> tag containing the character set specification can be added when specifying the XHTML document to load into an instance of EditLive!.
- Specifying the Character Set in the EditLive! Configuration File  
A <meta> tag containing character set specification can be nested in the <head> element of the EditLive! configuration file.

### Using the Document Load Time Property

The character set to be used within EditLive! can be specified in the document using the [Document](#) load time property. In order to specify the character set in this way, a <meta> tag must be included into the <head> of the document to be loaded into EditLive!.

#### Example

The following instantiation of EditLive! shows how to specify a character set of ASCII.

```
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

<script language="JavaScript">
  var editlivejava1;
  editlivejava1 = new EditLiveJava("ELJApplet1", "700", "400");
  editlivejava1.setConfigurationFile("sample_eljconfig.xml");
  editlivejava1.setDocument(escape("<html><head><meta http-equiv=\"Content-Type\" content=\"text/html; charset=ASCII\" /></head><body><h1>Body Text</h1></body></html>"));
  editlivejava1.show();
</script>
```

Note: If character encoding is specified by using the [setDocument Method](#), then this will be the final character encoding used. If an instance of EditLive! utilizes a configuration file that specifies character encoding, the character encoding will still be set to the value specified in the document loaded into the editor via the [setDocument Method](#).

### Specifying the Character Set In the EditLive! Configuration File

A meta tag explicitly specifying the character encoding can be entered into the configuration file for EditLive!.

#### Example

To specify ASCII character encoding, the following tag should be entered between the <head> tags of the configuration file as such:

```
<editLive>
  <document>
    <html>
      <head>
        <meta content="text/html; charset=ASCII" http-equiv="Content-Type" />
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Specifying the Output Character Set

The methods above specify how to define which character set is used to load and render the content in EditLive!. When extracting content from EditLive!, the default operation is to encode the outgoing content with the same character set specified for loading the content. If no character set has been specified for loading, the UTF-8 character set will be used for encoding outgoing content. Note the input vs output distinction; in such a scenario the system default would still be used for loading, which gives unpredictable results depending on user machine configurations.

Encoding characters with UTF-8 will correctly extract the content of EditLive! for most rendering character sets.

If you are experiencing problems regarding the content extracted from EditLive! not matching the content displayed in the editor, this may be due to an incompatibility between your rendering character set and the output character set. To explicitly set your output character set, use the [setOutputCharset Method](#).

## See Also

- [Character Sets Supported](#)
- [<meta> Configuration File Element](#)
- [setOutputCharset Method](#)

# Specifying Character Sets in the Swing SDK

Tiny EditLive! for Java Swing supports multiple character sets which allow it to be used in an international environment. Specifying the character set used by an instance of EditLive! for Java Swing defines which characters the editor can correctly render.

For a comprehensive list of the character sets supported by EditLive! for Java Swing, please see the [Character Sets Supported](#) article.

## Default Character Set

If a character set is not specified, the default character set for the user's JVM will be applied to the editor. This means the editor will exhibit different behaviour depending on the system language of the current user. On English Windows systems, the default character set is ISO8859\_1 (also known as CP1252). If, for example, you attempt to load content urlencoded as UTF-8 into the editor without specifying the UTF-8 characters set, a user loading the editor on English Windows would corrupt all unicode characters - eg "à" will be turned into "Ã¸". This corruption is not reversible.

## Specifying EditLive! for Java Swing's Character Set

The character set used by EditLive! for Java Swing can be specified by the following:

- Using the `setDocument()` method of `ELJBean`  
A `<meta>` tag containing the character set specification can be added when specifying the XHTML document to load into an instance of EditLive! for Java Swing.
- Specifying the Character Set in the EditLive! for Java Swing Configuration File  
A `<meta>` tag containing character set specification can be nested in the `<head>` element of the EditLive! for Java Swing configuration file.

### Using the `setDocument` method of `ELJBean`

The character set to be used within EditLive! for Java Swing can be specified in the document using the `setDocument()` method of `ELJBean`. In order to specify the character set in this way, a `<meta>` tag must be included into the `<head>` of the document to be loaded into EditLive! for Java Swing.

#### Example

The following instantiation of EditLive! for Java Swing shows how to specify a character set of ASCII.

```
ELJBean editlive = new ELJBean();  
  
...  
  
editlive.setDocument("<html><head><meta http-equiv=\"Content-Type\" content=\"text/html; charset=ASCII\" /></head><body><h1>Body Text</h1></body></html>");
```

If character encoding is specified by using the `setDocument()` method then this will be the final character encoding used. If an instance of EditLive! for Java Swing utilizes a configuration file that specifies character encoding, the character encoding will still be set to the value specified in the document loaded into the editor via the `setDocument()` method.

- Specifying the Character Set in the EditLive! Configuration File  
A `<meta>` tag containing character set specification can be nested in the `<head>` element of the EditLive! for Java Swing configuration file.

#### Example

To specify character encoding of ASCII, the following tag should be entered between the `<head>` tags of the configuration file as such:

```
<editLive>  
  <document>  
    <html>  
      <head>  
        <meta content="text/html; charset=ASCII" http-equiv="Content-Type" />  
        ...  
      </head>  
      ...  
    </html>  
  </document>  
  ...  
</editLive>
```

### Specifying the Character Set In the EditLive! for Java Swing Configuration File

A meta tag explicitly specifying the character encoding can be entered into the configuration file for EditLive!.

#### Example

To specify ASCII character encoding, the following tag should be entered between the `<head>` tags of the configuration file as such:

```
<editLive>
  <document>
    <html>
      <head>
        <meta content="text/html; charset=ASCII" http-equiv="Content-Type" />
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Specifying the Output Character Set

The methods above specify how to define which character set is used to load and render the content in EditLive! for Java Swing. When extracting content from EditLive! for Java Swing, the default operation is to encode the outgoing content with the same character set specified for loading the content. If no character set has been specified for loading, the UTF-8 character set will be used for encoding outgoing content. Note the input vs output distinction; in such a scenario the system default would still be used for loading, which gives unpredictable results depending on user machine configurations.

Encoding characters with UTF-8 will correctly extract the content of EditLive! for Java Swing for most rendering character sets.

If you are experiencing problems regarding the content extracted from EditLive! for Java Swing not matching the content displayed in the editor, this may be due to an incompatibility between your rendering character set and the output character set. To explicitly set your output character set, use the [setOutputCharset Method](#).

## See Also

- [Character Sets Supported](#)
- [<meta> Configuration File Element](#)

# Read Only Content and Custom Tags

- [Creating Read Only Content](#)
- [Using Custom Tags](#)

# Creating Read Only Content

Sections of content to be loaded into EditLive! can be marked as being read-only. EditLive! uses the *contenteditable* HTML attribute to indicate whether an element and its children are editable within EditLive!. If *contenteditable* is set to *false*, then the content of that element and its child elements become read-only.

The following example demonstrates how to create a read-only section within a document. The DIV with the text `<div>This is read only content.</div>` will be read-only.

```
<html>
  <body>
    <div> This content can be edited </div> <div contenteditable="false"> This is <b>read only</b> content. <
  /div> <div> This content is also able to be edited.</div>
  </body>
</html>
```

## Inheritance for Read-Only Content

When an element is marked as read-only, all its child elements are also created as read-only sections. However, the read-only property can be overwritten for specific child elements. This is achieved by explicitly setting the *contenteditable* attribute to *true* for that element. The following example demonstrates how to create a read-only parent element (a div) with an editable child element (a paragraph element).

```
<html>
  <body>
    <div> This content can be edited </div> <div contenteditable="false"> This content is not editable. <p
  contenteditable="true"> This content is contained within an editable child element</p>.<p> This content is not
  editable.</p></div><div> This content is also able to be edited. </div>
  </body>
</html>
```

## Setting an Entire HTML Document as Read-Only Using the Load-Time Properties

Using the *contenteditable* attribute of the `<body>` Configuration File element, developers can specify whether content is read-only. Developers can also use the [setReadOnly Method](#) to define whether the editor's contents are read-only.

See Also

- [<body> Configuration File Element](#)
- [setReadOnly Method](#)

# Using Custom Tags

EditLive! for Java Swing allows for the easy editing of HTML content by users. In some circumstances, however, content is not purely HTML and may contain custom tags or tags which are not defined in HTML. Examples are pages which include XML and Java Server Pages (JSP) tags. EditLive! for Java Swing recognizes the usage of such tags within a document, parses them, and renders them so that the user is aware of their existence.

## Custom Tag Identification

EditLive! for Java Swing identifies custom tags within HTML content placed in EditLive! for Java Swing. In the design view of EditLive! for Java Swing, custom tags are represented by an image of a yellow rectangle within which the tag name is displayed. Users can edit the custom tag by right-clicking on the tag and selecting **Edit Tag...** Please note that it is not possible to add extra HTML before or after the custom tag using this method.

When loading custom tags of the format `<custom>tag body</custom>` (i.e. tags with a body) into EditLive! for Java Swing, both the start and end tags will be recognized by EditLive! as custom tags and displayed as such. The body of the tag, in this case `tag body`, will be placed between the two custom tag representations and the user will be able to edit this content as they would any other content within EditLive! for Java Swing. `<SCRIPT>` tags, however, are an exception to this. These are displayed as a single script custom tag in EditLive! for Java Swing and the user must specifically edit the tag via the shortcut menu to make changes to the script.

## Custom Tag Support

EditLive! for Java Swing supports any custom tag that is correctly formatted. For instance the following tags are supported:

- `<custom>`
- `<custom tag="My Tag">`
- `<custom tag>`
- `</custom>`
- `<% asp script %>`
- `<? php script ?>`
- `<script language="JavaScript">JavaScript</script>`

The following tags will not work correctly:

- `<Unclosed Tag`
- `<? Incorrectly closed PHP >`
- `<% Incorrectly closed ASP >`
- `<script language="JavaScript">Unclosed JavaScript`

Notice that all of these unsupported tags are either unclosed or incorrectly closed. Ensuring that all tags are correctly closed will avoid such problems.

## Working with Custom Tags

Custom tags can be moved in the document just like any other element - including cutting, copying, pasting, and deleting. Users can also edit the custom tag by right clicking on the tag and selecting **Edit Tag...** from the shortcut menu if it is present. EditLive! for Java Swing maintains the case of custom tags throughout all operations. If EditLive! for Java Swing is set to indent HTML, then each line within a custom tag will be indented to the same level; however, if EditLive! for Java Swing is set to not indent HTML then the original indentation of the custom tag will be maintained.

## Unknown Custom Tags

EditLive! for Java Swing presents unknown custom tags (i.e. those not registered within EditLive! for Java Swing) as an image of a tag with a yellow background. Unknown tags are ignored by the EditLive! for Java Swing parser. For a custom tag to be affected by the EditLive! for Java Swing parser it must be registered with EditLive! as described in the following section.

## Registering Custom Tags

Developers can register custom tags with EditLive! for Java Swing so that they may be rendered and parsed in a specific manner. This enables developers to ensure that custom tags are used as either block, inline or empty tags and presented to users in a specific manner.

### Using CSS With Registered Custom Tags

Developers can specify the rendering of registered custom tags within EditLive! for Java Swing through the use of [Tiny \(Ephox\) CSS extensions](#). To register a custom tag in this manner it must be declared within either the external or embedded style sheet for the page. The custom tag is declared by including a specific CSS style for that tag and setting the display attribute for the tag to either block, inline or empty, which then determines how the tag is parsed.

The following example demonstrates how a custom tag called `MyTag` can be registered with EditLive! for Java Swing as a inline tag; the example also sets attributes relating to the appearance of the custom tag.

```
MyTag{
  display: inline;
  ephox-icon: url(http://www.server.com/icons/myicon.jpg);
  ephox-start-label: Custom Tag;
  ephox-end-label: /Custom Tag;
}
```



For more information on using Ephox CSS to render custom tags, see the [Using CSS Extensions to Render Custom Tags](#) article

### Using the Register Custom Tag method of the ELJBean class

This is only applicable to the EditLive! Swing SDK.

The ELJBean java class (which creates an instance of EditLive! for Java Swing in java) contains methods to register either block, inline or empty tags. When registering a tag using this method, the developer can specify the following values:

- *Start Label* - A string to represent the beginning of the custom tag
- *End Label* - A string to represent the end of the custom tag
- *Start icon* - An Icon to represent the beginning of the custom tag
- *End Icon* - An Icon to represent the end of the custom tag
- *ViewClickListener* - A class that will respond to the custom tag being clicked (this class could display a dialog or automatically change the custom tag's attributes)
- For more information, see the [Java Swing APIs](#) packaged with this SDK.

### See Also


- [Using CSS Extensions to Render Custom Tags](#)

# Equation Editor

- [Integrating the Tiny Equation Editor](#)

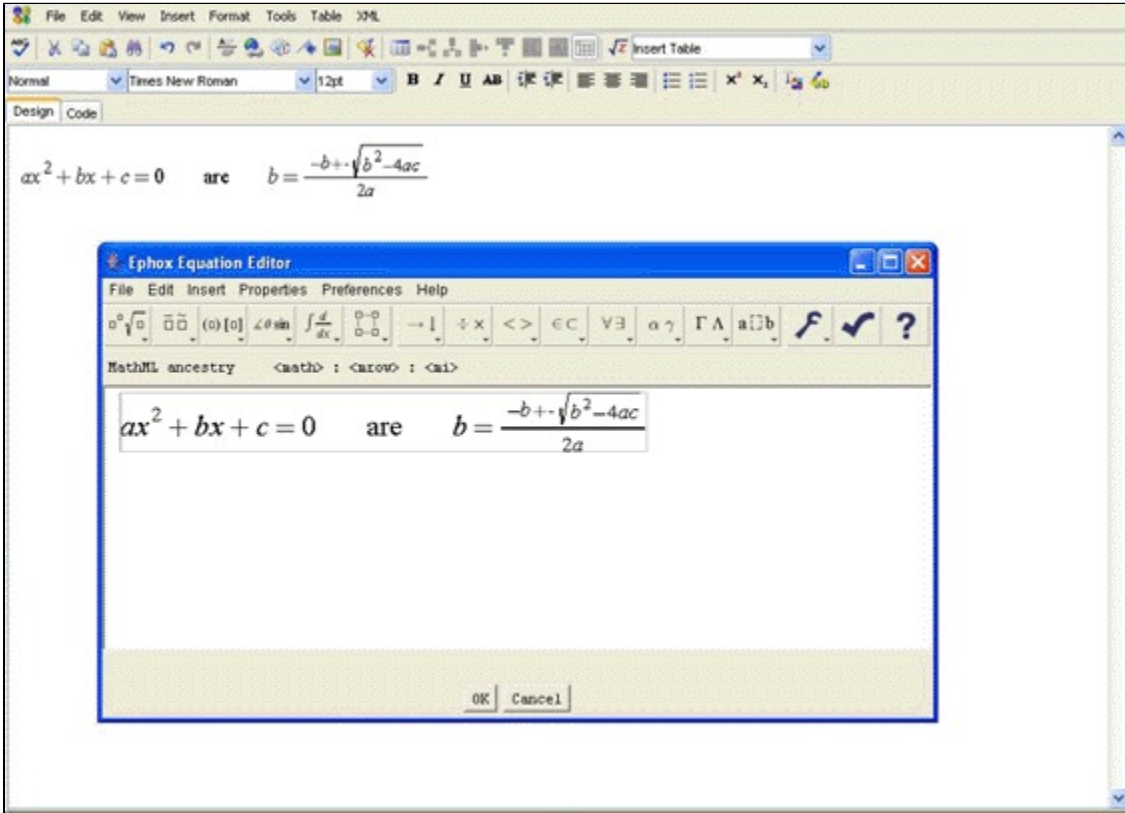
# Integrating the Tiny Equation Editor

In order for users to utilize the equation editing functionality, one of the following conditions must be met:

-  Users are still operating the editor in their 30 day trial period.
- An [Enterprise Edition](#) license has been installed for the editor. For more information on licensing see the [Licensing EditLive!](#) article.

The EditLive! Equation Editor enables content contributors to create and edit mathematical and scientific equations in EditLive!. The equation editor is available as an add-on and can either be purchased separately or in a package with EditLive! from Tiny. The equation editor enables the creation, editing and rendering of industry standard MathML content within EditLive!.

## EditLive! Equation Editor



## Installation

The EditLive! Equation Editor comes packaged automatically with EditLive!. Developers can activate the Equation Editor when using the default trial license packaged with EditLive!. In order to activate the Equation Editor in licensed versions of EditLive!, developers must specify licenses that have been generated with the Equation Editor specified as an inclusive feature.

For more information on ordering EditLive! licenses, please contact Tiny.

## Configuration

### Load Time Configuration

In cases where the EditLive! Equation Editor is to be used, EditLive! must be loaded with the [setUseMathML Method](#) called with the parameter *true*. This will cause EditLive! to download the source code required to run the equation editor. For more information on how to set the [setUseMathML Method](#), please see the [setUseMathML Method](#) information in the [Load Time Methods](#) section of this SDK.

### Configuration File Settings

In order for users to access the Equation Editor, the required links to the Editor have to be instantiated on the EditLive! Menu and Toolbar. For more information on the user interface commands available for use with the EditLive! Equation Editor, please see the Equation Editor Specific Menu and Toolbar Items section in the [Menu and Toolbar Item List](#) section of the EditLive! SDK.

The EditLive! Equation Editor can be specified to either store equations as mathml markup language or an **<img>** tag. For more information on specifying how EditLive! will store equations, see the **<mathml>** configuration element.

## See Also

- [setUseMathML Method](#)
- [<mathml> Configuration Element](#)
- [Menu and Toolbar Item List](#)

# Troubleshooting

- [Load Time Troubleshooting](#)
  - [Load Time Troubleshooting in the Applet](#)
  - [Load Time Troubleshooting in the Swing SDK](#)
- [Run Time Troubleshooting](#)
- [Minimizing an EditLive! Deployment](#)
- [Optimizing Load Time](#)
- [Managing Crashes](#)

# Load Time Troubleshooting

# Load Time Troubleshooting in the Applet

This article details common error messages displayed while initializing EditLive!. For each type of error message, this article details how to locate the source of the error and how to solve the problem.

## Accessing the Java Console

It will often be necessary to open the java console in order to understand exactly what is triggering an error in EditLive!. To open the java console, perform the following operations:

### Windows Platforms

In the system tray there should be a java icon. Right click on this icon and select 'Open Console'.

### Mac OS X

On the Macintosh, Java output is sent to the System console. This can be viewed using the Console application in the Application -> Utilities folder.

## XML Configuration File Error Messages

These error messages will be shown by the following display in EditLive!:

```
An error occured while parsing the XML configuration file
```

These error messages reflect a problem with one or more sections of the configuration file used to initialize EditLive!. When prompted with this error message, opening the Java console will reveal more information about the specific location and cause of the problem.

Common errors include:

### File Not Found Exception

This error will be shown by the following string in the Java console:

```
java.io.FileNotFoundException
```

This error refers to one of two conditions that could occur:

- The path specified in an EditLive! configuration file (using the [setConfigurationFile Method](#)) is incorrect. For more information on specifying the path of an EditLive! for Java configuration file, see the [Configuration File](#) load-time property documentation.
- The name of a specific EditLive! configuration file (using the [setConfigurationFile Method](#)) is incorrect. For more information on specifying the name of an EditLive! for Java configuration file, see the [Configuration File](#) load-time property documentation.

### Illegal Argument Exception

This error will be shown by the following string in the Java console:

```
java.lang.IllegalArgument
```

This error reveals that there is a syntax error in the EditLive! configuration file. This is usually due to a syntax error entered by a developer manually editing the configuration file.

## Character Encoding Error Message

This error message will be shown by the following display in EditLive!:

```
The required encoding for this document, XXXencodingMethodXXX, is unavailable on this system.
```

where *XXXencodingMethodXXX* represents the encoding method specified.

This error message reflects a problem with the character encoding method used for the content in EditLive!. Fixing this error will require locating where the character encoding was specified and specifying an acceptable character encoding method. The [Specifying Character Sets in the Applet](#) article details the three methods used to specify character encoding for EditLive! and its supported character sets.

## Licensing Error Messages

These messages will be displayed in a separate pop-up window from the EditLive! editor. Licensing information is specified in an EditLive! configuration file. For information on how to specify an EditLive! licence, see the [<license> Configuration File Element](#) article and the [Licensing EditLive!](#) article.

Common errors include:

### Account Not Found Error

This error will be shown by the following message in a pop-up window:

```
Licence error 203: Account not found
```

This error refers to an incorrect value being placed in the key attribute of a [<license>](#) configuration file element. Make sure the product key provided by Tiny is correctly entered into this attribute.

### License Domain Error

This error will be shown by the following message in a pop-up window:

```
Attempting to use an unlicensed domain
```

This error refers to an incorrect value being placed in the domain attribute of a [<license>](#) configuration file element. Make sure the organization's domain registered with Tiny is correctly entered into this attribute (commonly, the domain displayed by browsers in an organization is the domain name registered with Tiny).

## Applet Not Appearing Errors

You may experience a problem where EditLive! does not load; instead, a box appears on your page featuring a small red 'X'. In order to identify the cause for this error, open the java console.

### Class Not Found Exception

This error will be shown by the following string in the Java console:

```
java.lang.ClassNotFoundException: com.ephox.editlive.win.EditLiveJava
```

This indicates that the required EditLive! files were not able to be located to load the applet. This would be due to an incorrect value passed to the [setDownloadDirectory Method](#).

### QuickStart Applet Not Downloading the EditLive! .jar File

If you are using the `ephoxQuickStart()` function in Internet Explorer to preload the EditLive! .jar file, ensure the function is called within the document BODY.



# Load Time Troubleshooting in the Swing SDK

This article details common error messages displayed while initializing EditLive! for Java Swing. For each type of error message, this article details how to locate the source of the error and how to solve the problem.

## XML Configuration File Error Messages

These errors will cause the Java application to display the following error message where the instance of EditLive! for Java Swing should be located:

```
An error occurred while initializing. Please see Java console for details.
```

There are two different basic conditions. If the following pop-up dialog appears, the error indicates an incorrect path to the EditLive! for Java Swing configuration file was specified:

```
Error 3: Failed to load YOURCONFIG.xml
```

where *YOURCONFIG.xml* is the name of the EditLive! for Java Swing configuration you've specified.

If no pop-up dialog accompanies the error then there is a syntax error in the EditLive! for Java Swing configuration file. This is usually due to a syntax error entered by a developer manually editing the configuration file.

## Character Encoding Error Message

This error message will be shown by the following display in EditLive! for Java Swing:

```
The required encoding for this document, XXXencodingMethodXXX, is unavailable on this system.
```

where *XXXencodingMethodXXX* represents the encoding method specified.

This error message reflects a problem with the character encoding method used for the content in EditLive! for Java Swing. Fixing this error will require locating where the character encoding was specified and specifying an acceptable character encoding method. The [Specifying Character Sets in the Swing SDK](#) article details the three methods used to specify character encoding for EditLive! for Java Swing and its supported character sets.

## Licensing Error Messages

These messages will be displayed in a separate pop-up window from the EditLive! editor. Licensing information is specified in an EditLive! configuration file. For information on how to specify an EditLive! licence, see the [<license> Configuration File Element](#) article and the [Licensing EditLive!](#) article.

Common errors include:

### Account Not Found Error

This error will be shown by the following message in a pop-up window:

```
Licence error 203: Account not found
```

This error refers to an incorrect value being placed in the key attribute of a [<license>](#) configuration file element. Make sure the product key provided by Ephox is correctly entered into this attribute.

### License Domain Error

This error will be shown by the following message in a pop-up window:

```
Attempting to use an unlicensed domain
```

This error refers to an incorrect value being placed in the domain attribute of a [<license>](#) configuration file element. Make sure the organization's domain registered with Ephox is correctly entered into this attribute (commonly, the domain displayed by browsers in an organization is the domain name registered with Ephox).

## NoClassDefFound Error

Upon compiling you may encounter the following error message in the Java console:

```
Exception in thread "main" java.lang.NoClassDefFoundError: com/ephox/editlive/ELJBean
```

This error indicates that the required EditLive! for Java Swing files weren't correctly added to the classpath.

# Run Time Troubleshooting

This article details common errors encountered while operating EditLive! and solutions to solve these problems.

## Printing Errors

The printing architecture for EditLive! creates a new HTML web page containing the content of EditLive!. Through this new web page, users can then click the Print button to print their content based on the current printing settings configured in the user's browser.

### Printing Page Does Not Display

If the printing page does not display, check that the system EditLive! is operating on does not use webpage pop-up blockers.

### Page is Unable to Print

If the page is unable to print, an error exists either in the browser's printing settings or in the connection between the user's system and their printer. Users should consult their browser and operating system help files for more information.

## Image Insertion

Using an EditLive! configuration file, you can specify the location of a web page to list images the user can insert into an instance of EditLive!. For more information on configuring EditLive! to load this page, see the [Image Insertion Dialog's Browser Component](#) article and the `<imageBrowser>` configuration file element.

### Image Insertion Web Page Does Not Display Correctly

Unfortunately, while the page is being displayed in the Image Insertion dialog, JavaScript will not work. This may cause the appearance and functionality of the page to be compromised.

### Image Browser Dialog Does Not Appear

This error can occur if running EditLive! on Java Runtime Environment 1.4.2. This error occurs if a HTML form tag exists in the EditLive! document. The error is triggered if the HTML form tag does not include a target or action attribute. To ensure this error does not occur, include these attributes in any HTML form tag included in the EditLive! document.

## Spell Checking Errors

By default, the spell checking dictionary used with EditLive! is automatically configured to match the locale of the user's machine. However, if a developer wishes to specify a specific spell checking dictionary regardless of the user's locale, this can be done using the jar attribute of the `<spellCheck (Applet)>` configuration file element.

### Spell Checking Does Not Work

One of the most common reasons for spell checking failing is that the location of the dictionary jar file, specified in the `<spellCheck (Applet)>` configuration element, is incorrect. Check the location of the dictionary jar file again and verify this against the value specified in the `<spellCheck (Applet)>` configuration file element. Examples of EditLive! with functional dictionaries can be seen in the [Tutorials](#) section of this SDK.

## Accessibility Checking Errors

EditLive! allows users to check their document against the [W3C Accessibility Compliance Guidelines](#).

### Guideline WCAG 2 Check Name 3.3 Always Appears

This accessibility check refers changing all `<b>` and `<i>` tags in a document to `<strong>` and `<em>` respectively. When the **logicalEmphasis** attribute of the `<wysiwygEditor>` configuration file element is set to `false`, clicking the bold toolbar or menu item will insert `<b>` tags. If this attribute is set to `true`, clicking the bold link will create `<strong>` tags. If you are using accessibility checking, to avoid WCAG 2 Check Name 3.3 constantly appearing set the **logicalEmphasis** attribute to `true`.

## Saving Errors

### Browser Crashes on Form Submit

EditLive! can possibly crash your web browser if the following combination of code exists:

- EditLive!'s [setAutoSubmit Method](#) is called with `true` (this is the default value).
- A javascript call to `form.onSubmit()` is made to the html `<form>` containing EditLive!

To avoid crashing the browser, the following technique can be used:

Code your `onSubmit()` handler to return `-1`. Then, wherever `onSubmit()` is called, call `form.submit()` directly afterwards.

# Minimizing an EditLive! Deployment

It is possible to minimize a deployment of EditLive!. This can be especially useful when deploying EditLive! to Web servers with a limited amount of disk space available.

All the files required to deploy EditLive! to a Web server are included in the `INSTALL_HOME/webfolder/redistributables` directory of your EditLive! install where `INSTALL_HOME` is the directory where the SDK has been installed.

The redistributables directory contains two sets of files:

- EditLive! source files, found in the `.../redistributables` directory; and
- EditLive! dictionaries, found in the `.../redistributables/dictionaries` directory.

Note: When using EditLive! with ASP.NET or J2EE, the server control or tag library respectively must also be present on your Web server.

In the `redistributables/dictionaries` directory, it is possible to remove all the dictionaries apart from those being used with your implementation of EditLive!. The dictionary to be used with an instance of EditLive! is specified within the `<spellCheck (Applet)>` element of the configuration file.

In the `.../redistributables` directory, it is possible to deploy without the Java Runtime Environment (JRE) installer and instead force users to download the installer from Sun Microsystems. When deploying the JRE in this manner it must be ensured that clients have access to the Internet to download the installer. It should also be ensured that clients have the relevant access permissions to install the JRE on their machines.

In order to force users to download the JRE from Sun Microsystems the `setLocalDeployment Method` must be called with the parameter `false` when instantiating EditLive!.

# Optimizing Load Time

This document discusses several ways in which the load time of EditLive! can be optimized. These include preloading the Java Plug-in, configuring the instance of EditLive! via text embedded in the relevant Web page instead of using URLs, and deploying the Java Runtime Environment (JRE) in the manner best suited to your environment.

## Preloading the Java Plug-in

In order to run the EditLive! applet a browser must first load the Java Plug-in. The loading of the Java Plug-in occurs the first time a browser session seeks to use a Java applet, and the loading of the plug-in can take a noticeable amount of time. EditLive! includes functionality which allows for the preloading of the Java Plug-in for a browser session. This functionality can be added to any page within the relevant Web application to decrease the load time of EditLive! when it is eventually used. This functionality may be of most use when implemented on the user login page of a Web application, as it will decrease the load time for future uses of EditLive! within the Web application during the relevant session.

EditLive! provides the ability to preload the editor to decrease load time.

### Preloading EditLive!

Once EditLive! has loaded for the first time in a browser session, all subsequent loads of the editor will be much faster. Developers can capitalize on this functionality to load a hidden instance of EditLive! in a page before the editor is ever displayed to the end user.

An example of the preloading functionality, implemented in JavaScript, can be seen below. For more information please refer to the [setPreload Method](#).

#### Example

The following code would preload the JVM and EditLive! classes and then alert the user that EditLive! has finished loading by using the *preloadReturn* JavaScript callback function to create a JavaScript alert dialog.

```
<script language="javascript">
  ...
  var editlive1;
  editlive1 = new EditLiveJava("ELApplet1","1","1");
  editlive1.setDownloadDirectory("redistributables/editlivejava");
  editlive1.setConfigurationFile("editlivejava/sample_elconfig.xml");
  editlive1.setBody("&nbsp;");
  editlive1.setPreload("preloadReturn");
  editlive1.show();

  function preloadReturn(){
    alert("Preloading of the JRE and applet is complete.");
    // allow users to navigate to other webpages containing useable instances
    // EditLive! now that many of the classes associated with the editor have been cached in
    // the JVM.
  }

  ...
</script>
```

For Internet Explorer browsers, the Ephox [QuickStart](#) utility method already preloads EditLive!.

## Configuring EditLive! via the ConfigurationText Load-Time Property

When using an EditLive! configuration file to customize the EditLive! interface, EditLive! must make a HTTP request to the server to retrieve the relevant XML file. However, if the EditLive! configuration is set via an XML document which is embedded directly in the relevant page, EditLive! no longer has to make an extra HTTP request to the Web server and load time is decreased.

Embedding an EditLive! configuration document within the relevant Web page can be achieved by simply placing a URL encoded version of the XML configuration file onto the page. However, it is advisable to use the file reading capabilities of your server-side scripting language to read the relevant file directly from the Web server's file system into a temporary scripting variable on the relevant page and *then* load it into EditLive!. This is achieved via the [Configuration Text](#) load-time property.

#### Example

The following JavaScript code passes in an EditLive! configuration file document which will be used to customize EditLive!. The code uses the JavaScript *encodeURIComponent* function to encode the document; however, where possible, a server-side URL encoding method should be used to encode the XML text.

The EditLive! for Java configuration file XML document seen here is not complete. The XML text passed in must comply with the EditLive! Configuration File's DTD.

```
<script language="javascript">
  ...
  var editlive1;
  editlive1 = new EditLiveJava("ELApplet1","700","400");
```

```

editlive1.setDownloadDirectory("redistributables/editlivejava");
editlive1.setConfigurationText( encodeURI('<?xml version="1.0" encoding="UTF-8"?> <editLive>...');
editlive1.show();
...
</script>

```

## Setting the Document Styles

Setting the document styles via an external style sheet causes EditLive! to make a HTTP request to the relevant Web server. In order to avoid this the document styles can be set via the [setStyles Method](#).

### Example

The following code would set the initial style rules of the document within EditLive! for Java to be equal to:

```

body{ font-family: arial; background-color: white;}h1{ font-family: helvetica, arial, sans-serif; font-size:
16pt; font-style: normal; font-weight: bold;}p, td, th{ font-size: 12pt;}

```

This style information will be used to render the contents of EditLive! and will only be visible in the Code View.

The string passed to the **setStyles** property must be [URL encoded](#) or encoded using the JavaScript *encodeURI* function. The example below uses the JavaScript *encodeURI* function; however, where possible, a server side URL encoding method should be used.

```

<script language="javascript">
...
var editlive1;
editlive1 = new EditLiveJava("ELApplet1", "700", "400");
editlive1.setDownloadDirectory("redistributables/editlivejava");
editlive1.setConfigurationFile("editlivejava/sample_elconfig.xml");
editlive1.setBody(escape("<p>Initial contents of Ephox EditLive!</p>"));
editlive1.setStyles(escape('body{ font-family: arial; background-color: white;}
  h1{ font-family: helvetica, arial, sans-serif;
    font-size: 16pt; font-style: normal; font-weight: bold;}
  p, td, th{ font-size: 12pt;}'));
editlive1.show();
...
</script>

```

## Deploying the Java Runtime Environment

In order to use EditLive!, users must have the Java Runtime Environment (JRE) installed on their machine. If a user does not have the required version of the Java Runtime Environment installed on their machine it will be deployed automatically. In cases where the users of EditLive! are connected to a local intranet, it may be fastest to deploy the JRE from the local server. A copy of the JRE is provided with EditLive! for this purpose. To use this installer, simply set the local deployment property of EditLive! to *true*. The following example demonstrates how to achieve this with the [setLocalDeployment Method](#).

### Example

The following JavaScript code sets EditLive! to use the local copy of the JRE files from the server.

```

<script language="javascript">
...
var editlive1;
editlive1 = new EditLiveJava("ELApplet1", "1", "1");
editlive1.setDownloadDirectory("redistributables/editlivejava");
editlive1.setConfigurationFile("editlivejava/sample_elconfig.xml");
editlive1.setLocalDeployment(true);
editlive1.show();
...
</script>

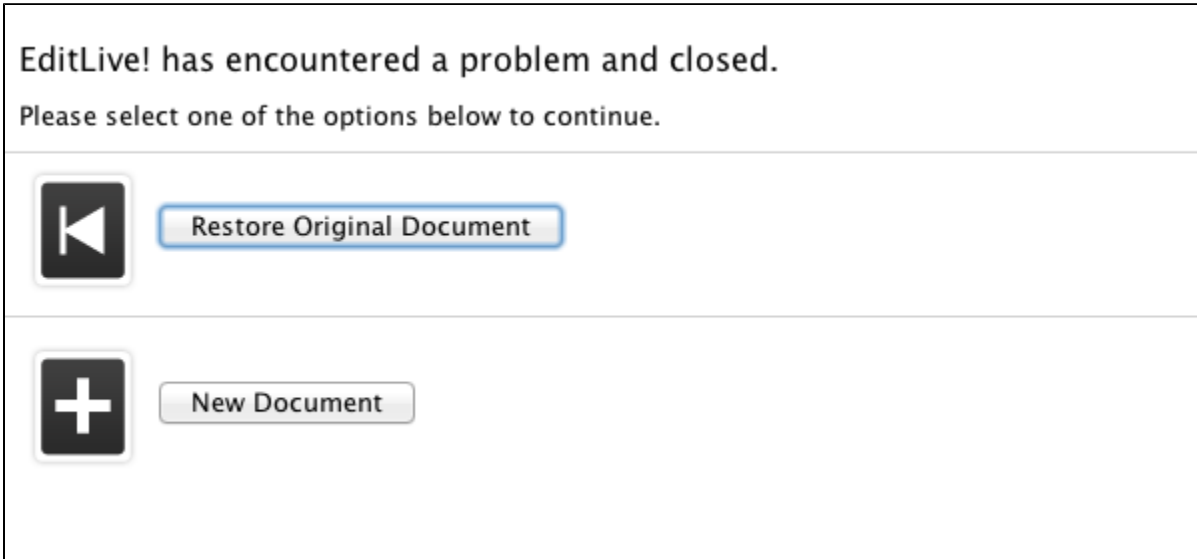
```

## See Also

- [setPreload Method](#)
- [setStyles Method](#)
- [setConfigurationText Method](#)
- [setLocalDeployment Method](#)
- [Optimizing Load Times Tutorial](#)

# Managing Crashes

When EditLive! runs out of memory or takes a long time to perform an operation such as paste, the editor aborts and displays the following:



**Restore original document** will load the content from when the user started editing in the current session (it loads content from use of `setBody()` or `setDocument()` at load time, not any calls to those methods after `show()`).

**New document** will wipe the slate clean and present the user with a blank document.

## Timeout

To configure the minimum timeout use the [MinCrashTimeout](#) property. This property enables the configuration of how long EditLive! will attempt to render a document before timing out and presenting the "Crash Screen" to the author. Specifically, the `MinCrashTimeout` property determines the amount of time to wait for individual calls to EditLive!'s rendering engine. This is very useful for use cases where it's likely large documents will be loaded. Some content that may fall into this category are long documents imported from Word and complex documents that contain many nested elements like tables and divs.

The default value for `MinCrashTimeout` is 10s. However, this *does not* mean that after 10s the editor will present the crash screen. This time is actually the amount of time that each individual operation that the editor performs will take. We cannot determine the actual number of operations that will be performed as it depends on the structure of the document.

As a guide, we recommend that users set their `MinCrashTimeout` to no less than **5s** and no more than **40s**. Setting the value too low will may cause standard documents to stop loading and present the crash screen. Setting the value too high will reduce the probability of the crash screen being displayed in a timely fashion and may cause the editor to appear to hang for extended periods of time.

## Customisation

The property [CrashAction](#) allows the crash screen to be customised.

A customised screen looks like this:

EditLive! has encountered a problem and closed.

Please select one of the options below to continue.



Restore Original Document



New Document



Custom Button

Runtime API methods will not work once the editor has crashed. This customisation is designed to allow the developer to manage the situation, for example by redirecting the user to a different page.



# Plugins

- [Creating and Using Plugins in the Applet](#)
- [Creating and Using Plugins in the Swing SDK](#)

# Creating and Using Plugins in the Applet

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

EditLive! allows developers to easily create and integrate plugins into the editor. Plugins can deliver additional functionality to EditLive! either through [Advanced API](#) extensions to the editor or through Javascript implementations of the run-time properties.

## Creating Plugins

Plugins are defined using the [Plugin XML](#) specifications. Plugin XML tells the editor meta information such as:

- The type of code the plugin will run (i.e. [Advanced API](#) Java code or Javascript).
- The location of the plugin code.
- Changes to the menus in the editor to allow the user to utilize new functionality provided in the plugin.
- Definitions on when the plugin should be loaded.

To create an EditLive! plugin, you need to perform the following steps:

1. Create the desired functionality as either an [Advanced API](#) implementation or a Javascript .js file.
2. Create [Plugin XML](#) to reference the functionality you created in step 1.
3. If required, create menu items in the Plugin XML to allow users to enable and utilize the functionality in your plugin.

This SDK provides two detailed tutorials on how to create plugins for use with EditLive!:

- [Simple Plugin Tutorial](#)
- [Creating Plugins Utilizing Advanced APIs Tutorial](#)

## Loading Plugins

EditLive! loads plugins by registering Plugin XML with the editor. Plugin XML can be registered with the editor via the following methods:

- [addPlugin Method](#)
- [addPluginAsText Method](#)
- `<Plugins>` Configuration File Element

# Creating and Using Plugins in the Swing SDK

EditLive! for Java Swing's [plugin](#) functionality is only supported with an EditLive! for Java Swing [Enterprise Edition](#) license.

EditLive! for Java Swing allows developers to easily create and integrate plugins into the editor. Plugins deliver additional functionality via Java classes that adhere to the specifications listed below.

## Creating Plugins

Plugins are defined using the [Plugin XML](#) specifications. Plugin XML tells the editor meta information such as:

- The location of the plugin code.
- Changes to the menus in the editor to allow the user to utilize new functionality provided in the plugin.
- Definitions on when the plugin should be loaded.

To create an EditLive! for Java Swing plugin, you need to perform the following steps:

1. Create the desired functionality via Java classes as specified below.
2. Create [Plugin XML](#) to reference the functionality you created in step 1.
3. If required, create menu items in the Plugin XML to allow users to enable and utilize the functionality in your plugin.

## Coding Java Classes to be Used as Plugins

When creating Java code to be loaded as an EditLive! for Java Swing plugin, the following steps need to be taken:

1. Your compiled Java classes need to be stored in a jar archive. This archive will be referenced via the [<advancedapis \(Applet\)>](#) Plugin XML.
2. The central class for your plugin (i.e. the Java class defined in the class attribute for the [<advancedapis \(Applet\)>](#) element) needs a constructor which accepts a single parameter: an instance of [ELJBean](#).

Loading Plugins

EditLive! for Java loads plugins by registering [Plugin XML](#) with the editor. Plugin XML can be registered with the editor via the following methods:

- [<plugins>](#) Configuration File Element

# HTML5 Support

EditLive! includes rendering and editing HTML5 media and semantic block elements. This document provides an overview of what EditLive! supports and how to turn HTML5 support on or off.

## Supported Elements

### HTML5 Semantic Block Elements

HTML5 semantic block elements operate within HTML in a similar way to DIVs. The key difference is that they provide semantic meaning to the content within them.

The following HTML5 semantic block elements are supported by the EditLive! Insert and Remove Section functionality:

#### Section Elements

```
<section/>
<header/>
<footer/>
<article/>
<aside/>
<nav/>
<figure/>
<figcaption/>
```

### Media Elements

The following Media elements are supported by the EditLive! Insert Media dialog:

#### Media Elements

```
<audio/>
<video/>
```


## Enabling/Disabling HTML5 Functionality

Settings to enable or disable HTML5 are described in the [Disabling HTML5 Features](#) documentation.

# Disabling HTML5 Features

HTML5 features may be unsupported or undesired in certain installation environments, particularly when targeting content against older web browsers that might not support HTML5. Within EditLive! HTML5 features can be disabled using the configuration items below.

Note

 Even when these features are disabled EditLive! will support rendering these HTML5 tags and users may still create content with these tags in the source code editor view.

## Removing HTML5 Audio and Video Features

The functionality to insert/edit HTML5 Audio and Video can be removed by setting the values of the *allowHtml5Audio* and *allowHtml5Video* attributes of the *multimedia* configuration element. This configuration is shown below:

### Audio + Video Tabs

```
<multimedia allowHtml5Audio="false" allowHtml5Video="false">
  ...
</multimedia>
```

This configuration will remove the Video and Audio tabs from the Insert Media dialog.

## Removing HTML5 Semantic Block Element Features

The functionality to insert/remove HTML5 section elements can be disabled by removing these items from the EditLive! configuration. The items that need to be removed are shown below:

### Section Operations

```
<menu name="ephox_insertmenu">
  <menuItem name="CreateSection" />
  <menuItem name="RemoveSection" />
</menu>
```

# Reference

- Load Time Methods
  - Load Time Properties Scope
  - JavaScript Constructor
  - addEditableSection Method
  - addJar Method
  - addPluginAsText Method
  - addPlugin Method
  - AppletSize Property (ASP.NET only)
  - closeOnFocusLost Method
  - Content Property (ASP.NET only)
  - EnableViewState Property (ASP.NET)
  - ID Property (ASP.NET only)
  - Init Method (ASP only)
  - InlineEditing Property (ASP.NET only)
  - InlineEditingCSS Property (ASP.NET only)
  - setAutoSubmit Method
  - setBaseUrl Method
  - setBody Method
  - setConfigurationFile Method
  - setConfigurationText Method
  - setCookie Method
  - setCrashAction Method
  - setDebugLevel Method
  - setDirection Method
  - setDirectionForEditableSection Method
  - setDisplayEditableMarker Method
  - setDocument Method
  - setDownloadDirectory Method
  - setEditableSectionCSS Method
  - setExpressEdit Method
  - setFocusOnLoad Method
  - setHead Method
  - setHeight Method
  - setHideButtonIconURL Method
  - setHideButtonText Method
  - setHttpLayerManager Method
  - setJREDownloadURL Method
  - setLocalDeployment Method
  - setLocale Method
  - setMinCrashTimeout Method
  - setMinimumJREVersion Method
  - setName Method
  - setOnInitComplete Method
  - setOutputCharset Method
  - setPreload Method
  - setReadOnly Method
  - setReturnBodyOnly Method
  - setShowButtonIconURL Method
  - setShowButtonText Method
  - setShowSystemRequirementsError Method
  - setStyles Method
  - setUseLiveConnect Method
  - setUseMathML Method
  - setUsername Method
  - setWidth Method
  - show Method
  - showAsButton Method
  - showInElement Method
  - setResizableSections
- Run Time Methods
  - closeActiveEditableSection Method
  - getBody Method
  - getCharCount Method
  - getContentForEditableSection Method
  - getDocument Method
  - getEditableSections Method
  - getSelectedText Method
  - getStyles Method
  - getWordCount Method
  - insertHTMLAtCursor Method
  - insertHyperlinkAtCursor Method
  - isDirty Method
  - openEditableSection Method
  - postDocument Method
  - removeEditableSection Method
  - setBackgroundMode Method

- setBody Method (Run Time)
- setContentForEditableSection Method
- setDocument Method (Run Time)
- setProperties Method
- uploadImages Method
- getContent Method
- performRaiseEvent method
- isEditableSectionDirty Method
- setIsDirty Method
- Utility Methods
  - quickStart Method
- Configuration File Elements
  - accessibilityChecks
  - action
    - action (Applet)
    - action (Swing SDK)
  - authentication
  - base
  - body
  - category
  - color
  - colorPalette
  - comboBoxItem
  - customComboBoxItem
    - customComboBoxItem (Applet)
    - customComboBoxItem (Swing SDK)
  - customMenuItem
    - customMenuItem (Applet)
    - customMenuItem (Swing SDK)
  - customTags
  - customToolBarButton
    - customToolBarButton (Applet)
    - customToolBarButton (Swing SDK)
  - customToolBarComboBox
  - document
  - doubleClickActions
  - editLive
  - ephoxLicenses
  - excellImport
  - head
  - html
  - htmlFilter
  - htmlImport
  - httpUpload
  - httpUploadData
  - httpPostData
  - hyperlink
  - hyperlinkList
  - hyperlinks
  - image
  - imageBrowser
  - imageDialog
  - imageList
  - images
  - inlineToolBar
  - inlineToolbars
  - license
  - link
  - mailtoLink
  - mailtoList
  - mathml
  - mediaSettings
  - menu
  - menuBar
  - menuItem
  - menuItemGroup
  - menuSeparator
  - meta
  - multimedia
  - otherLicenses
  - param
  - placesInDocumentList
  - plugins (config)
  - realm
  - repository
  - shortcutMenu
  - shrtMenu
  - shrtMenuItem
  - shrtMenuSeparator

- sourceEditor
- spellCheck
  - spellCheck (Applet)
  - spellCheck (Swing SDK)
- style
- submenu
- symbol
- symbols
- template
- templates
- textImport
- thesaurus
  - thesaurus (Applet)
  - thesaurus (Swing SDK)
- title
- toolbar
- toolbarButton
- toolbarButtonGroup
- toolbarComboBox
- toolbars
- toolbarSeparator
- trackChanges
- type
- types
- webdav
- webeqLicense
- wordImport
- wysiwygEditor
- services
- service
- contentLanguages
- language
- Plugin XML Elements
  - plugin
  - menu (plugin)
  - script
  - advancedapis
    - advancedapis (Applet)
    - advancedapis (Swing SDK)
- Java API
- Menu and Toolbar Items



# Load Time Methods

- Load Time Properties Scope
- JavaScript Constructor
- addEditableSection Method
- addJar Method
- addPluginAsText Method
- addPlugin Method
- AppletSize Property (ASP.NET only)
- closeOnFocusLost Method
- Content Property (ASP.NET only)
- EnableViewState Property (ASP.NET)
- ID Property (ASP.NET only)
- Init Method (ASP only)
- InlineEditing Property (ASP.NET only)
- InlineEditingCSS Property (ASP.NET only)
- setAutoSubmit Method
- setBaseURL Method
- setBody Method
- setConfigurationFile Method
- setConfigurationText Method
- setCookie Method
- setCrashAction Method
- setDebugLevel Method
- setDirection Method
- setDirectionForEditableSection Method
- setDisplayEditableMarker Method
- setDocument Method
- setDownloadDirectory Method
- setEditableSectionCSS Method
- setExpressEdit Method
- setFocusOnLoad Method
- setHead Method
- setHeight Method
- setHideButtonIconURL Method
- setHideButtonText Method
- setHttpLayerManager Method
- setJREDownloadURL Method
- setLocalDeployment Method
- setLocale Method
- setMinCrashTimeout Method
- setMinimumJREVersion Method
- setName Method
- setOnInitComplete Method
- setOutputCharset Method
- setPreload Method
- setReadOnly Method
- setReturnBodyOnly Method
- setShowButtonIconURL Method
- setShowButtonText Method
- setShowSystemRequirementsError Method
- setStyles Method
- setUseLiveConnect Method
- setUseMathML Method
- setUsername Method
- setWidth Method
- show Method
- showAsButton Method
- showInElement Method
- setResizableSections

# Load Time Properties Scope

## Overview

When implementing EditLive! within an application using the ASP load-time properties provided with EditLive!, it is important to note the scope of the properties once they are set. This section of the documentation details the scope of each load-time property. Properties with a global scope affect all instances of EditLive! within a page, while properties with a local scope affect only the instance of EditLive! they are set for.

## Distinguishing Between Local and Global Properties in ASP and ASP.NET

When using either the ASP load-time properties of EditLive!, local and global properties are set in separate objects or tags.

For ASP, global properties are set on the *EditLiveForJavaGlobal* object while local properties are set via the *EditLiveForJava* object.

For ASP.NET load-time properties, both local and global properties are set via the same tag. If multiple instances of EditLive! are declared in the same page, then the global properties set in the first instance of EditLive! take precedence over any others.

### Global Properties

- [AutoSubmit Property](#)
- [Cookie Property](#)
- [DebugLevel Property](#)
- [DownloadDirectory Property](#)
- [HttpLayerManager Property](#)
- [JREDownloadURL Property](#)
- [LocalDeployment Property](#)

Note: The [Init Method \(ASP only\)](#) must be called on the *EditLiveForJavaGlobal* object in the ASP load-time properties in order to load these values for all instances of EditLive! on the page.

### Local Properties

- [BaseURL Property](#)
- [Body Property](#)
- [ConfigurationFile Property](#)
- [ConfigurationText Property](#)
- [Document Property](#)
- [Head Property](#)
- [Height Property](#)
- [Locale Property](#)
- [Name Property](#)
- [OnInitComplete Property](#)
- [Preload Property](#)
- [ReturnBodyOnly Property](#)
- [Styles Property](#)
- [Width Property](#)

Note: The [Show Method](#) must be called on the *EditLiveForJava* object in the ASP load-time properties in order to load these values for the specific instance of EditLive! on the page.

# JavaScript Constructor

This method creates an instance of an Ephox EditLive! Javascript object.

This method only applies for EditLive! JavaScript integrations.

## Syntax

JavaScript

```
var editlive = new EditLiveJava(strName, width, height);
```

## Parameter

### strName

A unique string identifier for this instance of EditLive!.

This is a required parameter.

### width

This parameter will specify the width of the applet when displayed. This parameter can take the form of either:

- An integer representing the width of the applet in pixels (e.g. "200" for 200 pixels), or
- A percentage representing the width the applet consumes within the HTML element the applet is nested (e.g. "50%" for 50 percent).

This is a required parameter.

### height

This parameter will specify the height of the applet when displayed. This parameter can take the form of either:

- An integer representing the height of the applet in pixels (e.g. "200" for 200 pixels), or
- A percentage representing the height the applet consumes within the HTML element the applet is nested (e.g. "50%" for 50 percent).

This is a required parameter.

## Example

The following code creates an EditLive! object and assigns the identifier *editlive1* to the JavaScript variable. The object has a unique name of *ELApplet1*, a width of 700 pixels, and a height of 400 pixels.

```
var editlive1 = new EditLiveJava("ELApplet1", "700", "400");
```

## Remarks

When using percentage sizes, ensure that EditLive! is added to the page within an element that has an absolute size.

## See Also

- [show Method](#)
- [showInElement Method](#)

# addEditableSection Method

This property provides developers with a mechanism for editing multiple sections of HTML content in a webpage by utilizing a single instance of EditLive!.

For more information, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Using Inline Editing](#) tutorial.

## Syntax

JavaScript

```
editliveInstance.addEditableSection(divID[,textAreaID]);
```

## Parameters

### divID

The ID attribute for a HTML DIV. When a user clicks on this DIV, EditLive! will be loaded in its place, displaying the DIV's HTML contents. When the user clicks on any other DIV on the page registered via **addEditableSection**, the following occurs:

1. EditLive! is replaced by its original DIV.
2. The original DIV is populated with the contents seen in EditLive!.
3. EditLive! now appears in place of the last DIV clicked, populated with this DIV's HTML contents.

### textAreaID (optional)

The ID attribute for a HTML textarea, used internally by EditLive! to store and submit content.

This parameter is optional - it defaults to "divID\_contentArea".

If this element is not found, it is created.

If this element is found, and has no "name" attribute, the name is set to "divID\_contentArea".

As this element is used to submit content, its name determines the name of the HTTP form variable used to retrieve content.

If this element is found, its "value" attribute is used to load content. It takes precedence over the content inside the DIV. This is a more reliable method of loading content into the editable section than using the DIV content.

Like any HTML field, the textarea's "value" attribute must be HTML-encoded (not URL-encoded).

## Examples

The following code will load a webpage featuring a DIV element with a dashed border (with an ID attribute of "dashedDiv1"). When the user clicks this DIV, an instance of EditLive! will load in place of the DIV. Any HTML inside the DIV will then be copied into the instance of EditLive! to allow users to edit this HTML.

JavaScript

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="dashedDiv1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:
normal;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("dashedDiv1");
</script>
```

## Remarks

The **AddEditableSection** load-time property should only be invoked on a DIV after the DIV has been defined within the page. The best way to accomplish this is to define a function which uses the **AddEditableSection** load-time property within the HEAD element of the document, and then set the *onLoad* property of your BODY or FRAMESET element to call this function. In most cases, this will work unless there is a delay loading the entire content. In these cases, it may be best to call the **AddEditableSection** load-time property function after the last DIV has been defined. It is also worth noting that if the form the DIVs are within is submitted when the DIVs have not been added, then the content within the DIV will not be submitted back to the server. If it is important that the data is submitted back even if it is unaltered, then DIVs should be added within the body prior to the definition of any submit functionality or at least after each DIV has been defined.

The following example depicts how the **AddEditableSection** load-time property could be defined:

```
<html>
  <head>
```

```

<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<script language="Javascript">
    var editlivejs;
    function loadEditLive() {
        editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
        editlivejs.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejs.addEditableSection("div1");
    }
</script>
</head>

<body onload="loadEditLive()">
    <div id="div1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray;
white-space:normal;"><p>div content</p></div>
</body>
</html>

```

Any DIV element registered as an Inline Editing Section requires a fixed height. For example:

```

<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="dashedDiv1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:
normal;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("dashedDiv1");
</script>

```

If a fixed height is not specified for an Inline Editing Section DIV, clicking in the DIV will cause the Inline Editing Section to disappear from the page completely.

Finally, when using the **addEditableSection** API to integrate EditLive! into a webpage, the [Show](#) load-time property should not be invoked. When a user clicks a DIV registered with EditLive!, the editor will automatically be shown to the user in place of this DIV.

See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [setDisplayEditableMarker Method](#)

# addJar Method

Please see [addPlugin](#) and [addPluginAsText](#).

# addPluginAsText Method

EditLive!'s [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

This property is used to specify a plugin for use with EditLive!. See the [Creating and Using Plugins in the Applet](#) article in the [Developer Guide](#) for more information on using plugins.

This property differs from the [AddPlugin](#) load-time property in that the [Plugin XML](#) is passed through as a string rather than the location of the .xml file containing the plugin specifications.

Using [AddPluginAsText](#) will result in faster plugin loading times than [AddPlugin](#).

This property can be called any number of times for an instance of EditLive!.

## Syntax

JavaScript

```
editliveInstance.addPluginAsText(xmlString, [baseURL]);
```

## Parameters

### xmlString

The [Plugin XML](#) represented as a string.

The string passed to the JavaScript **addPluginAsText** property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript URL encoding functions do not fully comply with the URL encoding standard.

### baseURL

This parameter is optional.

Any relative URLs found in the [Plugin XML](#) (defined in **xmlString**) will be resolved against this base URL.

If no base URL is specified, relative URLs will resolve against the current URL for the page displaying the editor. For example, if EditLive! is being displayed on the page <http://myserver/edit/editPage.html>, the base URL will be <http://myserver/edit>.

## Examples

The following code would load a plugin based on the [Plugin XML](#) passed as the string argument.

JavaScript

```
editlivejs.addPluginAsText( "%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22US-AS..." );
```

Only specify *one* [<plugin>](#) element per call to **addPluginAsText**.

See Also

- [Plugin XML](#)
- [AddPlugin Load Time Property](#)

# addPlugin Method

EditLive!'s [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

This property is used to specify a plugin for use with EditLive!. For more information on plugins, please see the [Creating and Using Plugins in the Applet](#) article in the [Developer Guide](#) for this SDK.

This property can be called any number of times for an instance of EditLive!.

## Syntax

JavaScript

```
editliveInstance.addPlugin(xmlURL);
```

## Parameters

**xmlURL**

The URL specified location of the .xml file used to define the [Plugin XML](#).

## Examples

The following code would load a plugin based on the [Plugin XML](#) defined in the file *myPlugin.xml*.

JavaScript

```
editlivejs.addPlugin("myPlugin.xml");
```

## See Also

- [Plugin XML](#)
- [addPluginAsText Method](#)



# AppletSize Property (ASP.NET only)

This property specifies the height and width of the Ephox EditLive! applet when integrating the applet into an ASP.NET application.

## Syntax

ASP.NET

```
prefix:EditLiveJava AppletSize = intWidth, intHeight
```

## Parameters

### intWidth

This parameter will specify the width of the applet when displayed. The parameter can take the form of either:

- An integer representing the width of the applet in pixels (e.g. "200" for 200 pixels), or
- A percentage representing the width the applet consumes within the HTML element the applet is nested (e.g. "%50" for 50 percent).

The default value is 700.

### intHeight

This parameter will specify the height of the applet when displayed. The parameter can take the form of either:

- An integer representing the height of the applet in pixels (e.g. "200" for 200 pixels), or
- A percentage representing the height the applet consumes within the HTML element the applet is nested (e.g. "%50" for 50 percent).

The default value is 400.

## Examples

The following code would set the height of the EditLive! instance to 500 pixels and the width to 600 pixels.

ASP.NET Server Control

```
<elj:EditLiveJava  
  AppletSize = "600, 500"  
  ...  
>
```

## Remarks

Ephox recommends setting the height in pixels, as on Macintosh machines if these values are set as percentages and the Web browser is resized, EditLive! will not be resized with the window.

## See Also

- EditLive! [JavaScript Constructor](#)

# closeOnFocusLost Method

This property controls whether or not the active Inline Editing section is "closed" (replaced with a DIV) when focus moves out of the editor (e.g. if the user clicks on a text box). This property only applies to Inline Editing.

## Syntax

JavaScript

```
editliveInstance.setCloseOnFocusLost(bInCloseOnFocusLost);
```

ASP.NET - EditLiveJava Tag

```
prefix:EditLiveJava CloseOnFocusLost = bInCloseOnFocusLost
```

## Parameters

### **blnCloseOnFocusLost**

A boolean value indicating whether or not Inline Editing sections are closed once they lose input focus.

This value defaults to *false*.

## Examples

The following example demonstrates how to cause Inline Editing sections to automatically close when focus moves to another element.

JavaScript

```
var editlive_js;  
editlive_js = new EditLiveJava("ELApplet1", "700", "400");  
...  
editlive_js.setCloseOnFocusLost(true);
```

## Remarks

This can be used in Javascript as either a load-time or run-time property.

# Content Property (ASP.NET only)

This property is used to set/retrieve the content of a .net EditableSection control.

## Syntax

ASP.NET - EditLiveJava control

```
prefix:EditLiveJava Content="strContent"
```

ASP.NET - EditableSection control

```
prefix:EditableSection Content="strContent"
```

## Parameters

### strContent

A string specifying the initial content of the EditLive! applet.

The default value is an empty string.

## Examples

The following code would set the initial contents of EditLive! to be equal to *Initial contents*. The <P> and </P> tags will not be seen in the window as they will be parsed as HTML. However, these tags will be visible in the Code View.

ASP.NET EditableSection Control

```
<elj:EditableSection
  ...
  Content="<P>Initial contents</P>"
  ...
/>
```

This property can be databound to ASP.NET data sources.

# EnableViewState Property (ASP.NET)

This property specifies whether EditLive! maintains its view state between HTTP Posts. If set to *true*, EditLive! will be repopulated with the relevant data from the form submission. When set to *true* the data from the form submission will override any program settings of the EditLive! content. When set to *false*, the content of EditLive! is set to content specified programmatically or, if no content is specified programmatically, the content of EditLive! is set to the original content as specified in the instantiating server control tag.

This property only applies when using the ASP.NET Server Control, distributed with the ASP.NET SDK, to integrate EditLive!.

## Syntax

### ASP.NET

```
prefix:EditLiveJava EnableViewState = blnViewState
```

## Parameters

### blnViewState

A boolean indicating whether EditLive! should preserve the view state between HTTP Posts.

The default value is *true*.

## Examples

The following code will cause EditLive! to not preserve the view state between HTTP Posts.

### ASP.NET Server Control

```
<elj:EditLiveJava  
  ...  
  EnableViewState="false"  
  ...  
>
```

## Remarks

This property defaults to *true*.

When set to *true*, the data from the form submission will override any programmatic setting of the content of EditLive!.

When set to *false*, the content of EditLive! is set to content specified programmatically. If no content is specified programmatically, the content of EditLive! is set to the original content as specified in the instantiating server control tag.

# ID Property (ASP.NET only)

Specifies the ID for the ASP.NET controls. This is used by .net internally to identify the control. It is also used to generate the variable name of the EditLiveJava Javascript object instance, the "name" property of this instance, and the DIV and hidden textarea controls for [Inline Editing](#).

## Syntax

ASP.NET - EditLiveJava control

```
prefix:EditableSection ID = ID
```

ASP.NET - EditableSection control

```
prefix:EditableSection ID = ID
```

## Parameters

### ID

Unique identifier for the EditLiveJava control or InlineEditing section.

## Examples

The following code sets the ID for the EditableSection:

ASP.NET Server Control

```
<div id="div1" style="height: 550px;"></div>
<elj:EditLiveJava runat="server"
  ...
  ID="EditLiveJava1"
  InlineEditing="true"
  ..
/>
<elj:EditableSection runat="server"
  ...
  ID="EditableSection1"
  ...
/>
```

## Remarks

In order to use the **<elj:EditableSection>** tag and the **ID** property, you need to ensure you have the [InlineEditing](#) property for your EditLive! ASP.NET tag set to *true*.

## See Also

- [InlineEditing Property \(ASP.NET only\)](#)
- [Inline Editing ASP.NET Example](#)

# Init Method (ASP only)

This method initializes the Ephox EditLive! global object upon which it is called.

This method is only for use with the EditLive! ASP load-time properties.

## Syntax

Visual Basic Script

```
object.Init()
```

## Example

The following code declares a new EditLive! global object named *elglobal*, sets the required [DownloadDirectory property](#), and calls the **Init** method on it. This will initialize *elglobal* with the given [DownloadDirectory property](#) and the default [setLocalDeployment Method](#) value of *true*.

```
Dim elglobal
Set elglobal = New EditLiveForJavaGlobal
elglobal.DownloadDirectory = "../../../redistributables/editlivejava"
elglobal.Init()
```

## Remarks

Before the **Init** method is called on an EditLive! global object, the [DownloadDirectory property](#) of that object must be set.

## See Also

- [DownloadDirectory property](#)

# InlineEditing Property (ASP.NET only)

This property specifies whether [Inline Editing](#) is enabled for EditLive! when integrating the applet into an ASP.NET application.

## Syntax

ASP.NET

```
prefix:EditLiveJava inlineEditing = blnInlineEditing
```

## Parameters

### blnInlineEditing

This boolean parameter specifies whether [Inline Editing](#) is enabled.

The default value is *false*.

## Examples

The following code would enable [Inline Editing](#) mode for EditLive! in the ASP.NET integration.

ASP.NET Server Control

```
<elj:EditLiveJava  
  InlineEditing="true"  
  ...  
>
```

## Remarks

In order to specify which DIV elements you want to become [Inline Editing](#) sections, you will need to create separate `<elj:EditableSection >` tags with a [ID](#) attribute specified.

For more information see the [ID property](#) for the ASP.NET integration and the [Inline Editing in ASP.NET](#) example.

## See Also

- [Using Inline Editing](#)
- [ID Property \(ASP.NET only\)](#)
- [InlineEditingCSS Property \(ASP.NET only\)](#)
- [Inline Editing ASP.NET Example](#)

# InlineEditingCSS Property (ASP.NET only)

Removed in EditLive! 8.0



This property was removed in EditLive! 8.1.

See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [Inline Editing ASP.NET Example](#)
- [InlineEditing Property \(ASP.NET only\)](#)



# setAutoSubmit Method

This property specifies the way in which EditLive! behaves when the page is submitted. This affects how content is retrieved from EditLive!.

## Syntax

### Visual Basic Script

```
object.AutoSubmit = blnSubmit
```

### ASP.NET

This property is no longer present in the EditLiveJava ASP.NET control. The standard autosubmit mechanism is always off in ASP.NET and is replaced with alternative autosubmit functionality made specifically for ASP.NET.

### JavaScript

```
editliveInstance.setAutoSubmit(blnSubmit);
```

## Parameters

### blnSubmit

A boolean indicating if EditLive! should attach its content submission to the **onsubmit** function.

The default value is *true*.

## Examples

The following code would inform EditLive! to not attach its content submission to the **onsubmit** function.

### VBScript

```
editlive1.AutoSubmit = "false"
```

### JavaScript

```
editlivejs.setAutoSubmit(false);
```

## Remarks

When attaching its content submission to the **onsubmit** function, EditLive! populates a hidden field with its contents automatically rather than the developer calling for the contents explicitly. The name of the hidden field is contained within the same form as the EditLive! instance and is given the name that was specified by the developer when the EditLive! instance was created. For example, if the applet assigned the name *ELApplet1* was specified in the above example, EditLive! would store its contents in the hidden field named *ELApplet1*. This hidden field is then posted with the rest of the form data when the submit button is pressed. EditLive! automatically updates the hidden field by attaching itself to the form's **onsubmit()** handler. If there is already a function specified in the **onsubmit()** handler then this function will run after the hidden field has been updated. This means that you can still use the **onsubmit()** handler to run your own JavaScript functions. If you use another button/image/event to submit the form by calling **form.submit()**, the browser will not call the **onsubmit()** handler and EditLive! will not populate the hidden field with data. For this reason, please ensure you use **form.onsubmit()** to avoid this problem.

When deactivating the **onsubmit** functionality of EditLive! by setting the AutoSubmit property to *false*, the developer may wish to retrieve content from EditLive! using the [getDocument Method](#) provided in the EditLive! [Run Time Methods](#).

# setBaseURL Method

This property can be used to set the base URL used by EditLive! to resolve relative URLs (e.g. image URLs). The base URL property must be a URL for a virtual directory. The base URL property should be used in circumstances where it is impractical to set the `<base>` element of the EditLive! configuration file (e.g. when a single EditLive! configuration file is used within a system where EditLive! is used in multiple instances for editing pages with differing base URLs).

## Syntax

### Visual Basic Script

```
object.BaseURL = strBaseURL
```

### ASP.NET

```
prefix:EditLiveJava BaseURL = strBaseURL
```

### JavaScript

```
editliveInstance.setBaseURL(strBaseURL);
```

## Parameters

### strBaseURL

A string specifying the base URL to be used with this instance of EditLive!. The URL should map to a virtual directory. The base URL is used by EditLive! when resolving any relative URLs found within the editor.

## Examples

The following code would set the base URL for an instance of EditLive! to `http://www.yourserver.com/editor/`. This URL will be used when resolving all relative URLs in the EditLive! content and configuration file (e.g. URLs for images and links).

### VBScript

```
editlive1.BaseURL = "http://www.yourserver.com/editor/"
```

### ASP.NET Server Control

```
<elj:EditLiveJava
  ...
  BaseURL="http://www.yourserver.com/editor/"
  ...
/>
```

### JavaScript

```
editlive_js.setBaseURL("http://www.yourserver.com/editor/");
```

## Remarks

The base URL property must map to a virtual directory on a Web server and be a valid URL with a trailing `/`. For example, `http://www.yourserver.com/editor/` is a valid base URL while `http://www.yourserver.com/editor` is not.

Any value set in the `<base>` element of the EditLive! configuration file takes precedence over a value set through the base URL property. When using the base URL property to set the base URL it is recommended that you do *not* also set a value in the `<base>` element of the EditLive! configuration file.

## See Also

- [<base>](#) Configuration File Element

# setBody Method

This property specifies the initial document body contents of the Ephox EditLive! applet. When setting this property it should be noted that, when provided with a document, only information between the <BODY> and </BODY> tags will be retrieved and placed into the instance of EditLive!. This property is mutually exclusive with the [setDocument Method](#).

## Syntax

### Visual Basic Script

```
object.Body = strBody
```

### ASP.NET

This attribute no longer exists in the ASP.NET controls. Please use the [Content property](#) instead.

### JavaScript

```
editliveInstance.setBody(strBody);
```

## Parameters

### strBody

A string specifying the initial document body contents of the EditLive! applet.

The default value is an empty string.

## Examples

The following code would set the initial document body contents of EditLive! to be equal to *Initial Contents of Ephox EditLive!*. The <P> and </P> tags will not be seen in the window as they will be parsed as HTML. However, these tags will be visible in the Code View.

### VBScript

```
editlive1.Body = "<P>Initial contents of Ephox EditLive!</P>"
```

### JavaScript

The string passed to the JavaScript setBody property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript encodeURIComponent function does not fully comply with the URL encoding standard.

```
editlive_js.setBody(encodeURIComponent("<P>Initial contents of Ephox EditLive!</P>"));
```

The following code creates a <TEXTAREA>, named *bodyContents*, that will have its contents loaded into an instance of EditLive! via the SetBody function. The SetBody function will be associated with a HTML button. The name of the EditLive! for Java applet is *editlive\_js*.

```
<HTML>
<HEAD>
  <TITLE>EditLive! for Java JavaScript Example</TITLE>
  <!--Include the EditLive! for Java JavaScript Library-->
  <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript">
  </SCRIPT>
</HEAD>
<BODY>
  <FORM name = exampleForm>
    <P>EditLive! for Java contents will be loaded from here</P>
    <!--Create a textarea to load the applet contents from-->
    <P>
      <TEXTAREA name="bodyContents" cols="40" rows="10">
        <p>Content to be loaded</p>
      </TEXTAREA>
    </P>
```

```
<P>Click this button to set applet contents</P>
<P>
  <INPUT type="button"
    name="button1"
    value="Set Contents"
    onClick="editlive_js.SetBody(encodeURIComponent(document.exampleForm.bodyContents.value));">
</P>
<!--Create an instance of EditLive! for Java-->
<SCRIPT language="JavaScript">
  var editlive_js;
  editlive_js = new EditLiveJava("editlive",450 , 275);
  editlive_js.setDownloadDirectory("editlivejava");
  editlive_js.setLocalDeployment(false);
  editlive_js.setConfigurationFile("sample_elconfig.xml");
  editlive_js.show();
</SCRIPT>
</FORM>
</BODY>
</HTML>
```

## Remarks

The Body property is mutually exclusive with the [setDocument Method](#).

## See Also

- [setDocument Method](#)

# setConfigurationFile Method

This property specifies the URL at which the configuration file for EditLive! can be found. This file will customize the EditLive! interface. You may like to look at the [setConfigurationText Method](#) if you are considering dynamically generating EditLive! configuration files, but otherwise it is probably simpler to use this **ConfigurationFile** property.

## Syntax

Visual Basic Script

```
object.ConfigurationFile = strFileURL
```

ASP .NET

```
prefix:EditLiveJava ConfigurationFile = strFileURL
```

JavaScript

```
editliveInstance.setConfigurationFile(strFileURL);
```

## Parameters

### strFileURL

A string which is the URL for where the configuration file for this instance of EditLive! can be requested from. Ensure this string is correctly URL encoded.

## Examples

The following code specifies that EditLive! is to be loaded with the properties specified by the file *config.xml*, which can be found at *http://someserver/xmlconfig/config.xml*.

VBScript

```
editlive1.ConfigurationFile = "http://someserver/xmlconfig/config.xml"
```

ASP.NET Server Control

```
<elj:EditLiveJava
  ...
  ConfigurationFile = "http://someserver/xmlconfig/config.xml"
  ...
/>
```

JavaScript

```
editlivejs.setConfigurationFile("http://someserver/xmlconfig/config.xml");
```

## Remarks

Using the [setConfigurationText Method](#) to configure EditLive! results in a faster load time for the EditLive! applet than is achieved through the use of the **ConfigurationFile** property. If **ConfigurationFile** and [ConfigurationText](#) are both specified for an instance of the editor, [ConfigurationText](#) will take precedence.

If neither **ConfigurationFile** or **ConfigurationText** are specified, EditLive! will attempt to load the *sample\_eljconfig.xml* configuration file located in the directory specified by the [setDownloadDirectory Method](#).

Example

The following two code snippets would both specify the URL for the EditLive! configuration file as *../../redistributables/editlivejava/sample\_eljconfig.xml*.

```
<html>
  <body>
    <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
    <script language="Javascript">
      var editliveRef = new EditLiveJava("editlive", 700, 400);
      editliveRef.setDownloadDirectory("../../redistributables/editlivejava");
      editliveRef.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editliveRef.show();
    </script>
  </body>
</html>
```

```
<html>
  <body>
    <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
    <script language="Javascript">
      var editliveRef = new EditLiveJava("editlive", 700, 400);
      editliveRef.setDownloadDirectory("../../redistributables/editlivejava");
      editliveRef.show();
    </script>
  </body>
</html>
```

## See Also

- [setConfigurationText Method](#)
- [Instantiating the Applet](#)
- [EditLive! Configuration File Elements](#)

# setConfigurationText Method

This property or the [setConfigurationFile Method](#) (but not both) is required to be set for an Tiny EditLive! applet to run.

This property specifies the XML configuration text to be used by EditLive!. This text will customise the EditLive! interface. To find out about how to use EditLive! Configuration files, please read the article on [Manually Editing Configuration Files](#) and the EditLive! [Configuration File Elements](#) reference. The **ConfigurationText** property allows for the configuration of EditLive! via a string containing the configuration XML document to be used with EditLive!. Loading the configuration via the **ConfigurationText** property can reduce the load time of EditLive!. This can be most easily achieved by using server-side scripting to load the configuration file from the server's file system into a scripting string variable which can then be used when instantiating EditLive!.

## Syntax

### Visual Basic Script

```
object.ConfigurationText = strXMLText
```

### ASP.NET

```
prefix:EditLiveJava ConfigurationText = strXMLText
```

### JavaScript

```
editliveInstance.setConfigurationText(strXMLText);
```

## Parameters

### strXMLText

A string which contains the text of the XML configuration document for this instance of EditLive!.

## Examples

The following code would specify that Tiny EditLive! is to load with the given XML Configuration Text.

### VBScript

```
editlive1.ConfigurationText = "<?xml version='1.0' encoding='ISO-8859-1'?>  
<!DOCTYPE editLive SYSTEM '..'><editLive> <document>..."
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
  ...  
  ConfigurationText= "<?xml version='1.0' encoding='ISO-8859-1'?>  
<!DOCTYPE editLive SYSTEM '..'><editLive> <document>..."  
  ...  
</>
```

### JavaScript

```
editlivejs.setConfigurationText('%3C%3Fxml%20version%3D%221.0%22%3F%3E...');
```

The string passed to the JavaScript **setConfigurationText** property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript URL encoding functions do not fully comply with the URL encoding standard.

The XML document in the examples above is incomplete. It is given only as an example to aid understanding. The XML document passed to EditLive! via this method must be a complete XML configuration document.

## Remarks



Using the **ConfigurationText** property to configure EditLive! results in a faster load time than is achieved through the use of the [setConfigurationFile Method](#).

The **ConfigurationText** property is mutually exclusive with the [setConfigurationFile Method](#). You should provide either a URL to a configuration file, or pass in the configuration text in a String format.

## See Also

- [setConfigurationFile Method](#)
- [Instantiating the Applet](#)
- [EditLive! Configuration File Elements](#)

# setCookie Method

This property stipulates the name of the cookie to be used by Ephox EditLive!. Specifying a session's current cookie here allows EditLive! to retain the session information contained in this cookie. If the contents of EditLive! are sent to the server via a HTTP post, the contents of the cookie specified here are also passed. For more information on passing the contents of EditLive! to a server via HTTP post, see the [Retrieving Content From EditLive!](#) article.

## Syntax

### Visual Basic Script

```
object.Cookie = strCookie
```

### ASP.NET

```
prefix:EditLiveJava Cookie = strCookie
```

### JavaScript

```
editliveInstance.setCookie(strCookie);
```

## Parameters

### strCookie

A string value indicating the name of the Cookie. The value should be equivalent to a JavaScript value, for example *document.cookie*.

## Examples

The following code would set the Cookie property to *document.cookie*.

### VBScript

```
elglobal.Cookie = "document.cookie"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  Cookie = "document.cookie"  
...  
>
```

### JavaScript

```
editlivejs.setCookie(document.cookie);
```

## Remarks

The value used to set the **Cookie** property should be valid JavaScript. It is recommended that the value of *document.cookie* is used.

The value passed to this function will be evaluated as JavaScript. Thus, using the value of *document.cookie* with this function will result in the cookie for the HTML page being used by EditLive!.

# setCrashAction Method

This property allows a custom button to be added to the screen EditLive! displays if it crashes attempting to load a large document. For more information on how this customisation changes the screen please refer to [Managing Crashes](#).

This property requires EditLive! 7.6.0 or greater.

## Syntax

JavaScript

```
editliveInstance.setCrashAction(text, callback);
```

## Parameters

### text

A string used as the text on the "custom button" UI.

### callback

A JavaScript function reference that will be executed when the "custom button" is clicked.

## Examples

The following code will raise an alert dialog when the "custom button" button is clicked.

JavaScript

```
var crashFunction = function() {  
    alert("editor crashed");  
}  
editlivejs.setCrashAction("Custom Button", crashFunction);
```

# setDebugLevel Method

This property stipulates the level of debugging to be used when running EditLive!.

## Syntax

### Visual Basic Script

```
object.DebugLevel = strDebug
```

### ASP.NET

```
prefix:EditLiveJava DebugLevel = strDebug
```

### JavaScript

```
editliveInstance.setDebugLevel(strDebug);
```

## Parameters

### strDebugLevel

A string specifying the level of debugging to run EditLive! with. There are several distinct possible debug levels:

- *fatal*
- *error*
- *warn*
- *info*
- *debug*
- *http*

The default value is *info*.

## Examples

The following code would specify that the debug level is set to *debug*.

### VBScript

```
elglobal.DebugLevel = "debug"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  DebugLevel="debug"  
...  
>
```

### JavaScript

```
editlivejs.setDebugLevel("debug");
```

## Remarks

All information produced via the setting of the debug level is outputted to the Java console.

The following is a list of the possible debug levels in order of increasingly detailed output:

## fatal

This debugging level displays only error messages which prevent EditLive! from continuing, thus resulting in termination of the program.

## error

This debugging level displays error messages for cases in which EditLive! can continue despite the error. However, the current EditLive! operation will most likely fail due to the relevant error. This debugging level also displays all the information that would be displayed should the debugging level be set to *fatal*.

## warn

This debugging level displays messages indicating that an unexpected error has occurred and this may cause EditLive! to behave in an unexpected manner. However, the current EditLive! operation will most likely be completed successfully. This debugging level also displays all the information that would be displayed should the debugging level be set to *error*.

## info

This debugging level displays messages indicating that an event of some significance has occurred (e.g. a server has requested authentication details). EditLive! expects such events and deals with them accordingly. This debugging level also displays all the information that would be displayed should the debugging level be set to *warn*.

## debug

This debugging level displays any information which may be useful for debugging purposes. This debugging level also displays all the information that would be displayed should the debugging level be set to *info*.

## http

This debugging level displays communications using the HTTP client component of EditLive! (i.e. client server communications). This debugging level is most useful for tracking problems associated with HTTP connections. This debugging level also displays all the information that would be displayed should the debugging level be set to *debug*.

# setDirection Method

## Description

This property specifies the initial direction for the body contents of the Ephox EditLive! applet.

## Syntax

JavaScript

```
editliveInstance.setDirection(strDirection);
```

## Parameters

### strDirection

A string specifying the initial document body contents of the EditLive! applet. There are two possible language directions:

- *ltr*
- *rtl*

The default value is *ltr*.

## Examples

The following code would set the initial language direction of EditLive! to be right-to-left.

JavaScript

```
editlive_js.setDirection("rtl");
```

## Remarks

### rtl

This parameter will set the body to use right-to-left for the text direction.

### ltr

This parameter will set the body to use left-to-right for the text direction.

# setDirectionForEditableSection Method

This property specifies the initial direction for the contents of a specified Inline Editing Section (represented either by a DIV or an instance of EditLive!).

## Syntax

JavaScript

```
editliveInstance.setDirectionForEditableSection(strDivID, strDirection);
```

## Parameters

### strDivID

This parameter is required.

The ID attribute for the DIV registered as an Inline Editing Section with EditLive! (via the [addEditableSection Method](#)).

### strDirection

A string specifying the initial document body contents of the Inline Editing Section. There are two possible language directions:

- *ltr*
- *rtl*

The default value is *ltr*.

## Examples

The following code would set the initial language direction of an Inline Editing Section with an id of "div1" to be right-to-left.

JavaScript

```
editlive_js.setDirectionForEditableSection("div1", "rtl");
```

## Remarks

### rtl

This parameter will set the Inline Editing Section to use right-to-left for the text direction.

### ltr

This parameter will set the Inline Editing Section to use left-to-right for the text direction.

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)

# setDisplayEditableMarker Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

EditLive! provides a default rendering for users mousing over any DIV HTML element defined as an [Inline Editing Section](#). Prior to EditLive! 8.0, the **setDisplayEditableMarker** load-time property could be used to disable this default rendering for selecting Inline Editing Section. With the default mouse-over rendering disabled, you could code your own solution for how [Inline Editing Section DIVs](#) will render.

For more information on the Inline Editing Sections functionality of EditLive!, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Inline Editing Section](#) tutorial.

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)



# setDocument Method

This property specifies the initial document contents of the Tiny EditLive! applet.

When setting this property it should be noted that, when provided with a document, all the information from the document (everything between the <HTML> and </HTML> tags) will be placed inside the instance of EditLive!. This property is mutually exclusive with the [setBody Method](#).

## Syntax

### Visual Basic Script

```
object.Document = strDocument
```

### ASP.NET

This attribute no longer exists in the ASP.NET controls. Please use the [Content property](#) and the [Styles property](#) instead.

### JavaScript

```
editliveInstance.setDocument(strDocument);
```

## Parameters

### strDocument

A string specifying the initial document contents of the EditLive! applet.

The default value is an empty string.

## Examples

The following code would set the initial document contents of EditLive! to be equal to *Initial contents of Tiny EditLive!*

The HTML tags will not be seen in the window as they will be parsed as HTML. However, these tags will be visible in the *Code View* option of the EditLive! applet. Also, the title of the document (also only be visible in *Code View*) will be set to *Example*.

### VBScript

```
editlive1.Document = "<HTML><HEAD><TITLE>Example</TITLE></HEAD>  
<BODY><P>Initial contents of Tiny EditLive!</P></BODY></HTML>"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
    Document = "<HTML><HEAD><TITLE>Example</TITLE></HEAD>  
<BODY><P>Initial contents of Tiny EditLive!</P></BODY></HTML>"  
...  
</>
```

### JavaScript

```
editlivejs.setDocument(encodeURIComponent("<HTML><HEAD><TITLE>Example</TITLE></HEAD>  
<BODY><P>Initial contents of Tiny EditLive!</P></BODY></HTML>"));
```

The string passed to the JavaScript **setDocument** property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript URL encoding functions do not fully comply with the URL encoding standard.

## Remarks

The Document property is mutually exclusive with the [setBody Method](#).

# setDownloadDirectory Method

This property must be set for all EditLive! global objects. It specifies the directory in which the Ephox EditLive! source files can be found on the server.

## Syntax

### Visual Basic Script

```
object.DownloadDirectory = strDownloadDirectory
```

### ASP.NET

```
prefix:EditLiveJava DownloadDirectory = strDownloadDirectory
```

### JavaScript

```
editliveInstance.setDownloadDirectory(strDownloadDirectory);
```

## Parameters

### strDownloadDirectory

A string specifying the location of the EditLive! source files and JavaScript library.

## Examples

The following code would specify that the source files were in a directory called *redistributables/editlivejava* on the Web server.

### VBScript

```
elglobal.DownloadDirectory = "../../redistributables/editlivejava"
```

### ASP.NET Server Control

```
<elj:EditLiveJava
  ...
  DownloadDirectory="../../redistributables/editlivejava"
  ...
/>
```

### JavaScript

```
editlivejs.setDownloadDirectory("../../redistributables/editlivejava");
```

## Remarks

This property *must* be set when instantiating EditLive! when using the ASP or ASP.NET implementations of EditLive!.

When using the Javascript implementation of EditLive!, you don't need to define the **setDownloadDirectory** property if the source files for the editor are stored in the same location as the *editlivejava.js* referenced in the page.

### Example

The following two code snippets would both specify the EditLive! source files as located in the *../../redistributables/editlivejava/* directory relative to the webpage.

```
<html>
  <body>
    <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
```

```
        <script language="Javascript">
            var editliveRef = new EditLiveJava("editlive", 700, 400);
            editliveRef.setDownloadDirectory("../..../redistributables/editlivejava");
            editliveRef.setConfigurationFile("../..../redistributables/editlivejava/sample_eljconfig.
xml");
            editliveRef.show();
        </script>
    </body>
</html>
```

```
<html>
    <body>
        <script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"></script>
        <script language="Javascript">
            var editliveRef = new EditLiveJava("editlive", 700, 400);
            editliveRef.setConfigurationFile("../..../redistributables/editlivejava/sample_eljconfig.
xml");
            editliveRef.show();
        </script>
    </body>
</html>
```

# setEditableSectionCSS Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

When creating an instance of EditLive!, various methods can be used to define the CSS used by the editor. For a comprehensive list of the methods available for applying CSS to EditLive!, see the [Using CSS in the Applet](#) article in the [Developer Guide](#) for this SDK.

Any DIV registered with EditLive! as an [Inline Editing](#) Section can render its content using the same CSS as EditLive! - this is the default behavior. Prior to EditLive! 8.0, the **setEditableSectionCSS** load-time property could be used to enable or disable rendering [on an inline editing section](#) with the same CSS as EditLive!.

For more information on the Inline Editing functionality of EditLive!, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Using Inline Editing](#) tutorial.

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)

# setExpressEdit Method

**Select Edit works with TinyMCE 3.x only.**

When creating an instance of EditLive!, the TinyMCE Javascript editor can be loaded by default instead of the full Applet.

For more information on using Tiny supported TinyMCE, see the [Using TinyMCE](#) article in the [Developer Guide](#) for this SDK.

## Syntax

### JavaScript

```
editliveInstance.setExpressEdit(express);
```

## Parameters

### express

A value indicating whether EditLive! should show TinyMCE by default. There are four accepted values:

- **true** - TinyMCE will attempt to load first. If the platform and browser is not compatible with TinyMCE (see the [TinyMCE Compatibility Chart](#)), or if the content loaded into the editor contains [Track Changes](#) information or [Commenting](#), EditLive! will attempt to load. If Java is not installed on the client's platform or EditLive! is not compatible with the platform and browser being used, a textarea will be displayed.
- **false** - EditLive! will attempt to load. If Java is not installed on the client's platform or EditLive! is not compatible with the platform and browser being used, a textarea will be displayed.
- **"always"** - TinyMCE will attempt to load first. If the platform and browser is not compatible with TinyMCE (see the [TinyMCE Compatibility Chart](#)), or if the content loaded into the editor contains [Track Changes](#) information or [Commenting](#), a textarea area will be displayed.
- **"automatic"** - If Java is installed and the platform is supported by EditLive!, EditLive! will load. If EditLive! is unable to load, TinyMCE will attempt to load. If the platform and browser is not compatible with TinyMCE (see the [TinyMCE Compatibility Chart](#)), or if the content loaded into the editor contains [Track Changes](#) information or [Commenting](#), a textarea will be displayed.
- **"forceEditLive"** - EditLive! will attempt to load, regardless of whether the platform is compatible. If EditLive! is unable to load, an empty space will be displayed.

The default value is *false*.

## Examples

The following code will load an instance of TinyMCE if the client does not have Java installed.

### JavaScript

```
...  
  
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>  
<script language="JavaScript" src="../../redistributables/editlivejava/expressEdit/tinymce/jscripts/tiny_mce/  
/tiny_mce.js"/>  
  
...  
  
<script language="Javascript">  
    editlivejs = new EditLiveJava("ELApplet", 640, 400);  
    editlivejs.setExpressEdit("automatic");  
    editlivejs.show();  
</script>  
  
...
```

The following code will load an instance of TinyMCE, assuming the platform and browser being used is supported by TinyMCE.

```
...  
  
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>  
<script language="JavaScript" src="../../redistributables/editlivejava/expressEdit/tinymce/jscripts/tiny_mce/  
/tiny_mce.js"/>  
  
...  
  
<script language="Javascript">
```

```
editlivejs = new EditLiveJava("ELApplet", 640, 400);
editlivejs.setExpressEdit("always");
editlivejs.show();
</script>
...
```

## See Also

- [Using TinyMCE](#)
- [TinyMCE Known Issues](#)

# setFocusOnLoad Method

This property controls whether or not EditLive! gains input focus when it finishes loading.

## Syntax

### JavaScript

```
editliveInstance.setFocusOnLoad(bInFocusOnLoad);
```

### ASP.NET - EditLiveJava Tag

```
prefix:EditLiveJava FocusOnLoad = bInFocusOnLoad
```

## Parameters

### bInFocusOnLoad

A boolean value indicating whether or not EditLive! gains input focus when it finishes loading.

In Javascript, this value defaults to *false* for normal editing and *true* for inline editing.

The ASP.NET control defaults to *false* for both editing modes.

## Examples

The following example demonstrates how to cause EditLive! to automatically appear in a popout window with an associated button in the browser for showing and hiding the window.

### JavaScript

```
var editlive_js;  
editlive_js = new EditLiveJava("ELApplet1", "700", "400");  
...  
editlive_js.setFocusOnLoad(true);
```

# setHead Method

This property sets the content of EditLive! between the <HEAD> tags.

## Syntax

### Visual Basic Script

```
object.Head = strHead
```

### ASP.NET

This attribute no longer exists in the ASP.NET controls. Please use the Content property instead.

### JavaScript

```
editliveInstance.setHead(strHead);
```

## Parameters

### strHead

A string containing the text that will be the initial contents of the instance of EditLive! between the <HEAD> and </HEAD> tags.

The default value is an empty string.

## Example

The following code creates an instance of EditLive! and sets the initial contents of the document head to be <HEAD><TITLE>Example Head</TITLE><BASE href="http://www.ephox.com/" target="\_TOP"></HEAD>.

### VBScript

```
editlive1.Head = "<HEAD><TITLE>Example Head</TITLE><BASE href='http://www.ephox.com/' target='_TOP'></HEAD>"
```

### JavaScript

```
editlive_js.setHead(encodeURIComponent(' <HEAD><TITLE>Example Head</TITLE><BASE href="http://www.ephox.com/" target="_TOP"></HEAD> '));
```

The string passed to the JavaScript **setHead property** must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript URL encoding functions do not fully comply with the URL encoding standard.

## Remarks

When using the [setDocument Method](#), the document <HEAD> specified by the **Head** property overwrites any information set in the <HEAD> of the document specified by the [setDocument Method](#).

The information specified in the EditLive! XML Configuration Document in the <head> element overwrites any information specified for the document <HEAD> via either the [setDocument](#) or [setHead](#) load-time methods.



# setHeight Method

This property specifies the height of the Ephox EditLive! applet in pixels.

When creating an instance of EditLive! using JavaScript and ASP.NET, this property is not needed. The height of the applet is defined in the EditLive! [Java Script Constructor](#) for Javascript and by the [AppletSize property](#) for ASP.NET.

## Syntax

### Visual Basic Script

```
object.Height = intHeight
```

### ASP.NET - EditLiveJava control

```
prefix:EditLiveJava Height="unitHeight "
```

### ASP.NET - EditableSection control

```
prefix:EditableSection Height="unitHeight "
```

## Parameters

### intHeight

This parameter will specify the height of the applet when displayed.

This parameter can take the form of either:

- An integer representing the height of the applet in pixels (e.g. 200 for 200 pixels), or
- A percentage representing the height the applet consumes with-in the HTML element the applet is nested (e.g. 50% for 50 percent).

The default value is 400.

### unitHeight

This parameter will specify the height of the applet when displayed.

This parameter is a pixel or percentage value, e.g. "400px", "100%".

The default value is 400px.

## Examples

The following code would set the height of the EditLive! instance to 500 pixels.

### VBScript

```
editlive1.Height = 500
```

## Remarks

Ephox recommends setting the height in pixels, as on Macintosh machines if these values are set as percentages and the Web browser is resized, EditLive! will not be resized with the window.

See Also

- EditLive! [JavaScript Constructor](#)

# setHideButtonIconURL Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

This property specified the URL for the icon to be used as the hide button for EditLive! 7.6 and below. This function could only be used when displaying EditLive! as a button in the browser using the [ShowAsButton](#) load-time property.

## See Also

- [ShowAsButton Load Time Property](#)
- [setShowButtonText Method](#)
- [setShowButtonIconURL Method](#)
- [setHideButtonText Method](#)

# setHideButtonText Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

This property specified the text to be used with the hide button for EditLive! 7.6 and below. This function could only be used when displaying EditLive! as a button in the browser using the [ShowAsButton](#) load-time property.

## See Also

- [showAsButton Method](#)
- [setShowButtonText Method](#)
- [setShowButtonIconURL Method](#)
- [setHideButtonIconURL Method](#)

# setHttpLayerManager Method

This function has been removed in EditLive 9.1. The "default" Oracle/Sun layer is used for all HTTP connections.

This property specifies the way in which Ephox EditLive! manages HTTP interactions.

## Syntax

### Visual Basic Script

```
object.HttpLayerManager = httpLayer
```

### ASP.NET

```
prefix:EditLiveJava HttpLayerManager = httpLayer
```

### JavaScript

```
editliveInstance.setHttpLayerManager(httpLayer);
```

## Parameters

### httpLayer

A string specifying the HTTP connector responsible for managing HTTP requests generated from EditLive!. The accepted HTTP connectors are:

- *default*
- *apache*

The default value is *default*. The default HTTP upload manager is provided by Oracle's Java implementation.

## Examples

The following code would specify that the HTTP connector is set to *default*.

### VBScript

```
editlive1.HttpLayerManager = "default"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  HttpLayerManager="default"  
...  
>
```

### JavaScript

```
editlivejs.setHttpLayerManager("default");
```

## Remarks

The following is a list of the possible HTTP connectors used to manage HTTP interactions:


- **default**  
Oracle's URL class HTTP connector.

- **apache**  
HTTP connector created by Ephox using an Apache implementation.

# setJREDownloadURL Method

This property sets the location to download the required Java Runtime Environment from if it is needed on the client machine. This property can be used to specify a specific JRE for use with EditLive!.

This property *must* be set when deploying the JRE from a location other than Sun Microsystems' servers.

 This property only affects Internet Explorer.

## Syntax

### Visual Basic Script

```
object.JREDownloadURL = strJREURL
```

### ASP.NET

```
prefix:EditLiveJava JREDownloadURL = strJREURL
```

### JavaScript

```
editliveInstance.setJREDownloadURL(strJREURL);
```

## Parameters

### strJREURL

A string containing the location to download the Java Runtime Environment from if it is required to be installed.

## Examples

The following sets the JRE download URL to be *../JREDownload/j2re-1\_4\_1-windows-i586-i.exe*.

### VBScript

```
elglobal.JREDownloadURL = "../JREDownload/j2re-1_4_1-windows-i586-i.exe"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  JREDownloadURL = "../JREDownload/j2re-1_4_1-windows-i586-i.exe"  
...  
>
```

### JavaScript

```
editlivejs.setJREDownloadURL("../JREDownload/j2re-1_4_1-windows-i586-i.exe");
```

## Remarks

This may be a relative or absolute URL.

If a relative URL is specified then this will be relative to the URL of the page in which the EditLive! applet is embedded.

The **JREDownloadURL** property must specify the URL of a JRE installer executable.

# setLocalDeployment Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

This property stipulated where to download the Java Run Time Environment (JRE) from if it is not already installed on the client machine.

## See Also

- [setJREDownloadURL Method](#)

# setLocale Method

This property explicitly sets the locale that EditLive! should use for the interface translation, date formats, and other locale dependant properties.

## Syntax

### Visual Basic Script

```
object.Locale = strLocale
```

### ASP.NET

```
prefix:EditLiveJava Locale = strLocale
```

### JavaScript

```
editliveInstance.setLocale(strLocale);
```

## Parameters

### strLocale

Two letter ISO-369 compliant string representing the locale for the interface translation.

By default, if the interface translation is available for a client's locale, the interface appears in this translation unless explicitly set. If there is no translation available for a client's locale then the English interface translation is used.

## Examples

The following code would set the locale of EditLive! to German. This means that EditLive! will run with the German interface translation.

### VBScript

```
editlive1.Locale = "DE"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
  ...  
  Locale = "DE"  
  ...  
>
```

### JavaScript

```
editlivejs.setLocale("DE");
```

## Remarks

If this property is not set, the locale for EditLive! will be set according to the locale of the client's system properties.

If the set locale or the locale of the client's system is not supported by EditLive!, the English translation of the user interface is used by default.

Valid locales for EditLive! are:

- AR - Arabic
- CA - Catalan
- CS - Czech
- DA - Danish
- DE - German



- EL - Greek
- ES - Spanish
- FA - Farsi
- FI - Finnish
- FR - French
- HE - Hebrew
- HR - Croatian
- HU - Hungarian
- IT - Italian
- JA - Japanese
- KO - Korean
- NL - Dutch
- NB - Norwegian Bokmål
- PL - Polish
- PT - Brazilian Portuguese
- PT\_PT - European Portuguese
- RO - Romanian
- RU - Russian
- SK - Slovak
- SV - Swedish
- TH - Thai
- TR - Turkish
- ZH - Simplified Chinese
- ZH\_TW - Traditional Chinese

# setMinCrashTimeout Method

This property sets the minimum timeout for long operations that may result in EditLive! displaying the crash screen.

## Syntax

JavaScript

```
editliveInstance.setMinCrashTimeout(timeout);
```

## Parameters

### timeout

The minimum number of seconds to wait on any operation before displaying the crash screen.

The default value is 20.

## Examples

The following code will cause EditLive! to wait a minimum of three seconds for any editor action:

JavaScript

```
editlivejs.setMinCrashTimeout(3);
```

See Also

- [Managing Crashes](#) article

# setMinimumJREVersion Method

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

This attribute specified the minimum version of the Java Runtime Environment required to run EditLive! 7.6 and below. It should be noted that EditLive! should be used with JRE version 1.5 and above.

## See Also

- [setJREDownloadURL Method](#)

# setName Method

This property declares the name for the EditLive! applet. This name is used when posting the content from EditLive! as part of a HTML `<form>`. EditLive! posts its content in a form field which is named according to the value of the **Name** property. The style information from the EditLive! instance is posted in the field designated `NAME_styles` where `NAME` represents the value of the **Name** property.

Furthermore, the JavaScript object instantiated by the server-side scripting load-time and run-time properties is named `NAME_js` where `NAME` represents the value of the **Name** property. This JavaScript object is used when interacting with EditLive! via the JavaScript run-time properties.

This property is required to be set for an Ephox EditLive! for Java applet to run.

When creating an instance of EditLive! using JavaScript, this property is not needed. The name of the applet is defined in the EditLive! [JavaScript Constructor](#).

## Syntax

Visual Basic Script

```
object.Name = strName
```

ASP.NET (EditLiveJava control)

The `"Name"` and `"AppletName"` no longer exist on this control. Please use the `"ID"` property instead.

## Parameters

### **strName**

A string specifying a name for this applet.

## Examples

The following code would set the name of the EditLive! instance to `ELApplet1`.

VBScript

```
editlive1.Name = "ELApplet1"
```

## See Also

- EditLive! [JavaScript Constructor](#)

# setOnInitComplete Method

This property can be used to call a JavaScript function once EditLive! has finished loading. Once EditLive! has finished loading, the JavaScript function defined by the **OnInitComplete** property is used as a callback.

## Syntax

### Visual Basic Script

```
object.OnInitComplete = strCallBack
```

### ASP.NET

```
prefix:EditLiveJava OnInitComplete = strCallBack
```

### JavaScript

```
editliveInstance.setOnInitComplete(strCallBack);  
editliveInstance.setOnInitComplete(fnCallBack);
```

## Parameters

### strCallback

A string specifying the name of a JavaScript function to use as the callback function once EditLive! has finished loading.

### fnCallback

A Javascript Function to call once EditLive! has finished loading.

In EditLive! 8.0 and above, the callback is called much earlier than in previous versions. In EditLive! 8.0, the callback is called as soon as the EditLive! runtime API is available, which may be before the EditLive! applet is loaded. In previous versions, the callback was fired when the applet was loaded.

## Examples

The following code provides the JavaScript callback function which will set the body content of EditLive! once the applet has finished loading. The callback function is named *AppletLoaded*. This callback function will set the content of EditLive! to *Body content set at runtime*. The name of the EditLive! Applet JavaScript object is *ELApplet1\_js*.

The AppletLoaded function below uses the [setDocument Method \(Run Time\)](#) to set the body content of EditLive!. The string used to set the body content in this example is [URL encoded](#).

```
<script type='text/javascript'>  
  function AppletLoaded(){  
    ELApplet1_js.setDocument( "%3Cp%3ESet+content+at+runtime%3C%2Fp%3E" );  
  }  
</script>
```

The following example instantiates a version of EditLive! and assigns a function to be used as a callback once it has finished loading. The callback used in the example code is *AppletLoaded* which is described by the code above.

### VBScript

```
<%  
  ...  
  editlive1.Name = "ELApplet1"  
  editlive1.OnInitComplete = "AppletLoaded"  
  editlive1.Show()  
>
```

### ASP.NET Server Control

```
<elj:EditLiveJava Name="ELApplet1"  
  OnInitComplete="AppletLoaded"  
>
```

### JavaScript

```
var ELApplet1_js = new EditLiveJava("ELApplet1", "700", "400");  
ELApplet1_js.setOnInitComplete(AppletLoaded);
```

# setOutputCharset Method

This property specifies the output character set for EditLive!. The output character set defines how the contents of EditLive! are encoded when extracted from the editor (e.g using run-time properties such as the [GetDocument](#)).

If no output character set is specified, the output characters will match the specified rendering character set used by the editor. For more information on specifying the rendering character set for EditLive!, see the [Specifying Character Sets in the Applet](#) article in the [Developer Guide](#).

## Syntax

### Visual Basic Script

```
object.OutputCharset = strCharset
```

### ASP.NET

```
prefix:EditLiveJava OutputCharacterSet = strCharset
```

### JavaScript

```
editliveInstance.setOutputCharset(strCharset);
```

## Parameters

### strCharset

A string specifying the character set used by EditLive! when it outputs characters. There is a wide range of supported character sets. Supported character sets include:

#### ASCII

American Standard Code for Information Interchange

#### CP1252

Windows Latin-1

#### UTF8

Eight-bit Unicode Transformation Format

#### UTF-16

Sixteen-bit Unicode Transformation Format

#### ISO2022CN

ISO 2022 CN, Chinese

#### ISO2022JP

JIS X 0201, 0208 in ISO 2022 form, Japanese

#### ISO2022KR

ISO 2022 KR, Korean

#### ISO8859\_1

ISO 8859-1, Latin alphabet No. 1

#### ISO8859\_2

ISO 8859-2, Latin alphabet No. 2

ISO8859\_3

ISO 8859-3, Latin alphabet No. 3

ISO8859\_4

ISO 8859-4, Latin alphabet No. 4

ISO8859\_5

ISO 8859-5, Latin/Cyrillic alphabet

ISO8859\_6

ISO 8859-6, Latin/Arabic alphabet

ISO8859\_7

ISO 8859-7, Latin/Greek alphabet

ISO8859\_8

ISO 8859-8, Latin/Hebrew alphabet

ISO8859\_9

ISO 8859-9, Latin alphabet No. 5

ISO8859\_13

ISO 8859-13, Latin alphabet No. 7

ISO8859\_15

ISO 8859-15, Latin alphabet No. 9

SJIS

Shift-JIS, Japanese

A full list of character sets supported by version 1.4.2 of the Java Runtime Environment can be found on the [Sun Microsystems Java](#) website.

## Examples

The following example specifies that the characters used in EditLive! should be output using the UTF-8 character set.

### VBScript

```
editlive1.OutputCharset = "UTF-8"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
  ...  
  OutputCharacterSet = "UTF-8"  
  ...  
>
```

### JavaScript

```
elinstance.setOutputCharset("UTF-8");
```

## Remarks

The ASCII character set will be used if no output character set is specified.



# setPreload Method

This property can be used when preloading EditLive!. Once EditLive! has finished loading, the JavaScript function defined by the **Preload** property is used as a callback.

This feature disables plugins so that dialogs are not shown on load.

## Syntax

### Visual Basic Script

```
object.Preload = strCallBack
```

### ASP.NET

```
prefix:EditLiveJava Preload = strCallBack
```

### JavaScript

```
editliveInstance.setPreload(strCallBack);
```

## Parameters

### strCallBack

A string specifying the name of a JavaScript function to use as the callback function once EditLive! has finished loading.

## Examples

The following code provides the JavaScript callback function which will display an alert dialog once EditLive! has finished loading. The callback function is named *preloadReturn*.

```
<script language="javascript">
  function preloadReturn(){
    alert("EditLive! has finished preloading.");
  }
</script>
```

The following example instantiates a version of EditLive! and assigns a function to be used as a callback once it has finished loading. The callback used in the example code is *preloadReturn* which is described by the code above. The example below instantiates an applet which is not visible; thus, it may be used in cases where the applet is to be preloaded to decrease load times for future instances, but not visible.

### VBScript

```
<%
...
editlive1.Name = "ELApplet1"
editlive1.Width = "1"
editlive1.Height = "1"
editlive1.ConfigurationFile = "sample_elconfig.xml"
editlive1.Body = "<p>&nbsp;</p>"
editlive1.Preload = "preloadReturn"
editlive1.Show()
%>
```

### ASP.NET Server Control

```
<elj:EditLiveJava Name="ELApplet1"
  Width="1"
```

```
Height="1"
ConfigurationFile="sample_elconfig.xml"
Body="<p>&nbsp;</p>"
Preload="preloadReturn"
/>
```

## JavaScript

```
var editlivejs;
editlivejs = new EditLiveJava("ELApplet1", "1", "1");
editlivejs.setConfigurationFile("sample_elconfig.xml");
editlivejs.setDocument(escape("<p>&nbsp;</p>"));
editlivejs.setPreload("preloadReturn");
```

## Remarks

The **Preload** property can be used to assist with the preloading of EditLive!. This can improve the performance of EditLive! within a Web application. Preloading causes the browser's XML Plug-In and the EditLive! classes to be loaded.

It is recommended that, when preloading EditLive!, you set the height and width of the EditLive! applet so they are both one pixel. This will ensure that the EditLive! applet is not visible on the page.

Preloading EditLive! can be performed on any page within a Web application.

When preloading the applet, no plugins specified for use with the editor will be loaded.

# setReadOnly Method

This property stipulates whether the user of EditLive! is able to edit the contents of the editor.

## Syntax

### Visual Basic Script

```
object.ReadOnly = strRead
```

### ASP.NET

```
prefix:EditLiveJava ReadOnly = strRead
```

### JavaScript

```
editliveInstance.setReadOnly(strRead);
```

## Parameters

### strRead

A string which specifies if the user is able to edit the contents of EditLive!. If strReturn is *true*, then only the body is returned. If strReturn is *false*, then the entire document is to be returned.

There are only two possible values for this attribute: *true* and *false*.

## Examples

The following code would specify that the user of EditLive! is unable to edit the editor's contents.

### VBScript

```
editlive1.ReadOnly = "true"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
  ...  
  ReadOnly = "true"  
  ...  
>
```

### JavaScript

```
editlive_js.setReadOnly("true");
```

# setReturnBodyOnly Method

This property stipulates what content to retrieve from Tiny EditLive!. It affects the following:

- [getContent Method](#)
- [getContentForEditableSection Method](#)
- content posted when `autoSubmit` is set to `true`

For more information on how EditLive! instances can attach their content to a form submission see the [Retrieving Content From EditLive!](#) and [setAutoSubmit Method](#) articles.

When set to `true`, EditLive! returns all content between the `<body>` and `</body>` tags, not including the `<body>` and `</body>` tags. When set to `false`, EditLive! returns all content between the `<HTML>` and `</HTML>` tags. When set to `auto`, EditLive! automatically predicts what content to return.

## Syntax

### Visual Basic Script

```
object.ReturnBodyOnly = strReturn
```

### ASP.NET

```
prefix:EditLiveJava ReturnBodyOnly = strReturn
```

### JavaScript

```
editliveInstance.setReturnBodyOnly(strReturn);
```

## Parameters

### strReturn

A string which specifies if only the body, or if the entire document is to be returned, or if EditLive! is to automatically detect what is to be returned. If `strReturn` is `true`, then only the body is returned. If `strReturn` is `false`, then the entire document is to be returned. If `strReturn` is `auto`, then EditLive! is to automatically detect what is to be returned (see below).

There are only three possible values for this attribute: `true`, `false`, and `auto`.



The default value is `true`.

## Examples

The following code would specify that only the body is to be returned.

### VBScript

```
editlive1.ReturnBodyOnly = "true"
```

### ASP.NET Server Control

```
<elj:EditLiveJava
...
  ReturnBodyOnly = "true"
...
/>
```

### JavaScript

```
editlive_js.setReturnBodyOnly("true");
```

## Remarks

The default of this property is *true*. Hence, if this attribute is not included in your code, by default only the body will be returned. The body consists of everything between, but not including, the <body> tags.

When setting this property to *auto*, EditLive! will perform in the following way:

- If the content is set using the [setDocument Method](#), a whole document is returned
- If the content is set using the [setBody Method](#), only the content within the <body></body> tag is returned
- If the content is set for an Inline Editing section using the [setContentForEditableSection Method](#), the content returned depends on the content set
  - if a whole document is set, a whole document is returned; if a fragment is set, a fragment is returned
- If the content is set for an Inline Editing section using the related textarea, the content returned depends on the content set.
- If the content is set for an Inline Editing section by populating the Inline Editing element (not recommended), the content returned depends on the content set.

# setShowButtonIconURL Method

This property specified the URL for the icon to be used with the show button for EditLive!. This function could only be used when displaying EditLive! as a button in the browser using the [showAsButton Method](#).

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

## See Also

- [showAsButton Method](#)
- [setShowButtonText Method](#)
- [setHideButtonIconURL Method](#)
- [setHideButtonText Method](#)

# setShowButtonText Method

This property specified the text to be used with the show button for EditLive!. This function could only be used when displaying EditLive! as a button in the browser using the [showAsButton Method](#).

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

## See Also

- [showAsButton Method](#)
- [setShowButtonIconURL Method](#)
- [setHideButtonIconURL Method](#)
- [setHideButtonText Method](#)

## setShowSystemRequirementsError Method

This property specified the way in which Tiny EditLive! reacted when a client machine did not match the system requirements needed to run EditLive!.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.



# setStyles Method

This property specifies the styles rules to be contained in the <STYLE> element within the <HEAD> section of the document in EditLive!.

## Syntax

### Visual Basic Script

```
object.Styles = strStyles
```

### ASP.NET

```
prefix:EditLiveJava Styles = strStyles
```

### JavaScript

```
editliveInstance.setStyles(strStyles);
```

## Parameters

### strStyles

A string specifying the style rules to be loaded into a <STYLE> element in the <HEAD> section of the document within the EditLive! applet.

## Examples

The following code would set the initial style rules of the document within EditLive! to be equal to

```
"p.msonormal { font-size: 12pt; margin: 0cm 0cm 0pt; font-family: 'times new roman' } h1 } font-weight: bold; font-size: 12pt; margin-left: 0cm; color: black; margin-right: 0cm; font-family: helvetica )}"
```

. This style information will be used to render the contents of EditLive! and will only be visible in the Code View.

### VBScript

```
editlive1.Styles = "p.msonormal { font-size: 12pt; margin: 0cm 0cm 0pt; font-family: 'times new roman' } h1 { font-weight: bold; font-size: 12pt; margin-left: 0cm; color: black; margin-right: 0cm; font-family: helvetica }"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  Styles="p.msonormal { font-size: 12pt; margin: 0cm 0cm 0pt; font-family: 'times new roman' }  
h1 { font-weight: bold; font-size: 12pt; margin-left: 0cm; color: black; margin-right: 0cm; font-family:  
helvetica }"  
...  
>
```

### JavaScript

```
var editlive1;  
editlive1 = new EditLiveJava("ELApplet1", "600", "400");  
editlive1.setBody(encodeURIComponent("<p>Initial contents of Ephox EditLive!</p>"));  
editlive1.setStyles(encodeURIComponent("p.msonormal { font-size: 12pt; margin: 0cm 0cm 0pt; font-family: 'times new  
roman' }  
h1 { font-weight: bold; font-size: 12pt; margin-left: 0cm; color: black; margin-right: 0cm; font-family:  
helvetica }"));
```

---

The string passed to the JavaScript `setStyles` property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript URL encoding functions do not fully comply with the URL encoding standard.

## See Also

- [setBody Method](#)
- [setDocument Method](#)

# setUseLiveConnect Method

LiveConnect is a method supported by many browsers for communicating between Java applets and Javascript.

The **setUseLiveConnect** method was used to enable EditLive! to communicate with the webpage using LiveConnect. This encompasses communication such as the [Run Time Methods](#) and the [Auto-Submission](#) architecture for the editor.

With the **setUseLiveConnect** method called with *true*, communication speeds between Javascript and the applet were improved.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

## setUseMathML Method

This property specified whether EditLive! should make the EditLive! Equation Editor functionality available to create, edit, and render mathematical and scientific equations in MathML. For more information on how to install and configure the EditLive! Equation Editor see the EditLive! [Equation Editor](#) article.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

### See Also:

- EditLive! [Equation Editor](#)

# setUserName Method

This property is used to set the current user's name. This username is appended to all changes made when the user edits the contents of EditLive! when Track Changes is enabled. For more information on Track Changes see the [Getting Started With Track Changes](#) article in the [Developer Guide](#) section of this SDK.

## Syntax

### Visual Basic Script

```
object.Username = strUserName
```

### ASP.NET

```
prefix:EditLiveJava UserName = strUserName
```

### JavaScript

```
editliveInstance.setUsername(strUserName);
```

## Parameters

### strUserName

A string specifying the name for the current user of EditLive!.

## Examples

The following code would set the username appended to all track changes as *'Ephox User'*.

### VBScript

```
editlive1.Username = "Ephox User"
```

### ASP.NET Server Control

```
<elj:EditLiveJava  
...  
  UserName="Ephox User"  
...  
>
```

### JavaScript

```
editlive_js.setUsername("Ephox User");
```

## Remarks

If two users enter the same name (e.g. "John Smith"), the changes made by these two separate users will render as if the changes were made by the same user. In order for each user to have their changes uniquely tracked and rendered, each user will need to enter a unique username.

## See Also

- [Getting Started With Track Changes](#)

# setWidth Method

This property specifies the width of the Ephox EditLive! applet in pixels.

When creating an instance of EditLive! using JavaScript or ASP.NET, this property is not needed. The height of the applet is defined in the EditLive! [JavaScript Constructor](#) for Javascript and by the [AppletSize property](#) for ASP.NET.

## Syntax

### Visual Basic Script

```
object.Width = intWidth
```

### ASP.NET - EditLiveJava control

```
prefix:EditLiveJava Width="unitHeight"
```

### ASP.NET - EditableSection control

```
prefix:EditableSection Width="unitHeight"
```

## Parameters

### intWidth

This parameter will specify the width of the applet when displayed. This parameter can take the form of either:

- An integer representing the width of the applet in pixels (e.g. 200 for 200 pixels), or
- A percentage representing the width the applet consumes within the HTML element the applet is nested (e.g. 50% for 50 percent).

The default value is 700.

### unitHeight

This parameter will specify the height of the applet when displayed. This parameter can be a pixel or percentage value, e.g. "400px", "100%".

The default value is 700px.

## Examples

The following code would set the width of the EditLive! instance to 800 pixels.

### VBScript

```
editlive1.Width = 800
```

## Remarks

Ephox recommends setting the width in pixels, as EditLive! will not be resized with the window on Macintosh machines if these values are set as percentages and the Web browser is resized.

# show Method

This method displays the Ephox EditLive! instance in the Web browser.

This method only applies for EditLive! ASP and JavaScript integrations.

## Syntax

### Visual Basic

```
object.Show()
```

### JavaScript

```
editliveInstance.show();
```

## Examples

The following code sets only the required properties of an EditLive! applet and then displays the applet within the Web page.

### VBScript

```
editlive1.Show()
```

### JavaScript

```
var editlive_js;  
editlive_js = new EditLiveJava("ELApplet1", "700", "400");  
...  
editlive_js.show();
```

## Remarks

Before this method is called, all the required properties of an EditLive! instance object must be set.

## See Also

- [showAsButton Method](#)

# showAsButton Method

This method displayed the Tiny EditLive! instance object in the Web browser as a button. When the parameter was set to *true*, EditLive! would automatically open in a popout window; clicking an associated button in the browser would hide the window. When set to *false*, EditLive! displayed as a button which must be clicked in order for the popout window containing EditLive! to appear.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. Calling this method will fail with an exception.

## See also:

- [setShowButtonText Method](#)
- [setShowButtonIconURL Method](#)
- [setHideButtonIconURL Method](#)
- [setHideButtonText Method](#)



# showInElement Method

This method displays the EditLive! instance as a child of the specified control.

[AutoSubmit](#) must be turned off in order to use **showInElement**. These two features are not compatible.

## Syntax

JavaScript

```
editliveInstance.showInElement(parentID);
```

## Parameters

### parentID

A string specifying the ID of the parent element to insert the editor into.

JavaScript

```
<div id="parentElement"></div>

<script type="text/javascript">
    var editlive_js;
    editlive_js = new EditLiveJava("ELApplet1", "700", "400");
    ...
    editlive_js.showInElement("parentElement");
</script>
```

## Remarks

This property is used as a replacement to [show](#). This method is **not** to be used when invoking the editor via the [addEditableSection Method](#).

Before this method is called, all the required properties of an EditLive! instance object must be set.

This method works better than `show()` in AJAX environments.

# setResizableSections

This property controls whether EditLive! will resize the editable area when content changes. It is recommended that this only be enabled for inline editing with the tabs turned off.

For more information on inline editing see the [Using Inline Editing](#) article.

Important



It is recommended that a doctype be used on your webpage to ensure that Internet Explorer does not use IE5 Quirks mode. Inline sections may not resize correctly in Internet Explorer if the page is using IE5 Quirks mode.

See the [W3C Recommended list of Doctype declarations](#) for information on doctype declarations that can be used.

## Syntax

JavaScript

```
editliveInstance.setResizableSections(bResizable);
```

## Parameters

### **bResizable**

A boolean value indicating whether or not EditLive! will resize the editable area when content changes.

This value defaults to *false*.

## Examples

The following code demonstrates a page with two resizable sections and one non-resizable section. It requires that three divs with the ID's "div1", "div2" and "div3" exist on the page. Only the first and third inline sections will resize when content changes.

JavaScript

```
var editlive_js;  
editlive_js = new EditLiveJava("ELApplet1", "700", "400");  
...  
editlive_js.setResizableSections(true);  
editlive_js.addEditableSection("div1");  
editlive_js.addEditableSection("div3");  
editlive_js.setResizableSections(false);  
editlive_js.addEditableSection("div2");
```

# Run Time Methods

The run-time properties for EditLive! are Javascript functions allowing the user to interact with the editor. The run-time properties can only be invoked once the editor has completed loading.

# closeActiveEditableSection Method

## Description

This method closes the active Inline Editing editable section - closing the applet and restoring the content to the Inline Editing DIV. If no sections are open, this method does nothing.

For more information, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Using Inline Editing](#) tutorial.

## Syntax

JavaScript

```
editliveInstance.closeActiveEditableSection();
```

## Example

The following code adds an editable section to the page, and provides a button that closes the section.

JavaScript

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="dashedDiv1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:
normal;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("dashedDiv1");
</script>

<input type="button" value="Close Active Section" onclick="editlivejs.closeActiveEditableSection();" />
```

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [openEditableSection Method](#)
- [removeEditableSection Method](#)

# getBody Method

This function retrieves the contents of the EditLive! applet. It will retrieve all contents which lie between the <BODY> tags.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.getBody([jsFuncnt], [blnUploadImages]);
```

## Parameters

### jsFuncnt

This parameter is optional.

The javascript function which receives the EditLive! applet contents.

### blnUploadImages

This parameter is optional.

This is a boolean which indicates whether images should be uploaded to the server when this method is called. The uploading of images will occur immediately before the content is retrieved.

The default value is *false*.

## Returns

The EditLive applet contents.

## Synchronous Example (EditLive 8.1+)

Calling **getBody** synchronously returns the <body> contents of the editor as a string. This example retrieves the contents of the editor and then saves the retrieved contents to a <textarea> with the name *bodyContent*. Images will be uploaded to the server when **getBody** is called as *blnUploadImages* is set to true. The name of the EditLive applet is *editlive\_js*.

```
document.exampleForm.bodyContent.value = editlive_js.getBody(true);
```

## Asynchronous Example (EditLive 8.0+)

When calling **getBody** asynchronously you must provide a JavaScript function as the callback parameter for the **getBody** method. In this example the *retrieveBody* callback function receives the contents of the editor as a string. Images will be uploaded to the server when **getBody** is called as *blnUploadImages* is set to *true*. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getBody("retrieveBody", true);
```

The *retrieveBody* function receives the contents of the editor and then saves the retrieved contents to a <textarea> with the name *bodyContents*.

```
function retrieveBody(src){
    document.exampleForm.bodyContent.value = src;
}
```

## Remarks

When uploading locally stored images to the relevant Web server for an instance of EditLive!, ensure that the *blnUploadImages* parameter is set to *true* when calling the **getBody** method.

## See Also

- [Retrieving Content From EditLive!](#)



# getCharCount Method

This function obtains a count of the number of characters present within the EditLive! applet. It counts the number of characters which are contained within controls on the current view.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
int editliveInstance.getCharCount([jsFunc]);
```

## Parameters

### jsFunc

This parameter is optional.

The JavaScript function which receives the character count obtained from the EditLive! applet. This parameter can also be a string representation of the javascript function.

## Returns

Number of characters in the document.

Note: this is the number of characters in the actual text of the document, not the number of characters in the HTML representation of the document.

## Example Synchronous (EditLive 8.1+)

Calling **getCharCount** synchronously returns the the character count as a string. This example retrieves the character count and then displays it in a JavaScript alert dialog. The name of the EditLive applet is *editlive\_js*.

```
var charCount = editlive_js.getCharCount();
```

## Example Asynchronous (EditLive 8.0+)

When calling **getCharCount** asynchronously you must provide a JavaScript function as the callback parameter for the **getCharCount** method. In this example the *charCountAlert* callback function receives the character count as a string. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getCharCount('charCountAlert');
```

The *charCountAlert* function receives the character count as a string and displays it in a JavaScript alert dialog.

```
function charCountAlert(count){
    alert('Character Count: ' + count);
}
```

# getContentForEditableSection Method

This method retrieves the contents of a specified Inline Editing Section. Before content is returned, this function ensures all local images in the content are uploaded by calling [uploadImages](#).

For more information on the Inline Editing functionality for EditLive!, see the [Using Inline Editing](#) article.

In EditLive! 8.0, the **getContentForEditableSection** function will return a HTML fragment or a HTML document depending on the content used to populate the Inline Editing section.

In EditLive! 8.1 and above, the **getContentForEditableSection** function returns content according to the [setReturnBodyOnly](#) setting.

For information on populating Inline Editing sections see the [Using Inline Editing](#) article.

## Syntax

### JavaScript

```
editliveInstance.getContentForEditableSection(divID);
```

## Parameters

### divID

This parameter is required.

The ID attribute for the DIV registered as an inline editing section with EditLive! (via the [addEditableSection Method](#)).

## Example

The following code creates a webpage featuring a DIV. This DIV is registered with EditLive! as an inline editing section. The *Display Content* button will display the current HTML for the inline editing section.

If the user clicks the DIV, EditLive! will load in its place. Clicking the *Display Content* button would then load the current contents of the editor to the screen.

If the user does not click the DIV, pressing *Display Content* will load the contents of the DIV itself to the screen.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"/>
    <SCRIPT language="JavaScript">
      function displayEditableSectionContent(){
        var contentRef = editlive_js.getContentForEditableSection('div1');

        alert("Editable Section Content: " + editlive_js.getContentForEditableSection('div1'));
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>Click the section below to edit its contents.</P>
      <DIV id="div1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:normal;"><p>default editable section content</p></DIV>
      <P>Click the following button to display the content for the editable section: <INPUT type="button" name="divButton" value="Display Content" onClick="displayEditableSectionContent();" >
    </P>
    </FORM>
    <SCRIPT language="JavaScript">
      var editlive_js = new EditLiveJava("editlive", "100%", "100%");
      editlive_js.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
      editlive_js.addEditableSection('div1');
    </SCRIPT>
  </BODY>
</HTML>
```



## See Also

- [Using Inline Editing Tutorial](#)
- [addEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [setEditableSectionCSS Method](#)
- [setDisplayEditableMarker Method](#)

# getDocument Method

This function retrieves the entire contents of the EditLive! applet.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.getDocument([jsFunct],[blnUploadImages]);
```

## Parameters

### jsFunct

This parameter is optional.

The JavaScript function which receives the retrieved EditLive! applet contents.

### blnUploadImages

This parameter is optional.

This is a boolean which indicates whether images should be uploaded to the server when this function is called. The uploading of images will occur immediately before the content is retrieved.

The default value is *false*.

## Returns

The EditLive applet contents.

## Example Synchronous (EditLive 8.1+)

Calling **getDocument** synchronously returns the contents of the editor as a string. This example retrieves the contents of the editor and then saves the retrieved contents to a <textarea> with the name *documentContent*. Images will be uploaded to the server when **getDocument** is called as *blnUploadImages* is set to *true*. The name of the EditLive applet is *editlive\_js*.

```
document.exampleForm.documentContent.value = editlive_js.getDocument(true);
```

## Example Asynchronous (EditLive 8.0+)

When calling **getDocument** asynchronously you must provide a JavaScript function as the callback parameter for the **getDocument** method. In this example the *retrieveDocument* callback function receives the contents of the editor as a string. Images will be uploaded to the server when **getDocument** is called as *blnUploadImages* is set to *true*. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getDocument('retrieveDocument',true);
```

The *retrieveDocument* function receives the contents of the editor and then saves the retrieved contents to a <textarea> with the name *documentContents*.

```
function retrieveDocument(src){
    document.exampleForm.documentContent.value = src;
}
```

## Remarks

When uploading locally stored images to the relevant Web server for an instance of EditLive!, ensure that the *blnUploadImages* parameter is set to *true* when calling the **getDocument** method.

# getEditableSections Method

This function returns an array of ID attributes for each DIV registered with EditLive! as an Inline Editing Section.

For more information on the Inline Editing functionality for EditLive!, see the [Using Inline Editing](#) article.

## Syntax

JavaScript

```
editliveInstance.getEditableSections();
```

## Example

The following code creates a webpage featuring three DIV HTML elements. Each DIV is registered with EditLive! as an inline editing section. The **Display Editable Sections** button will display the ID attributes for each DIV registered with EditLive! as an inline editing section.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"/>
    <SCRIPT language="JavaScript">
      function displayEditableSectionDIVIDs(){
        var editableSectionArray = editlive_js.getEditableSections();

        for(i = 0; i < editableSectionArray.length; i++) {
          alert("Editable Section DIV ID: " + editableSectionArray[i]);
        }
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>Click the section below to edit it's contents.</P>
      <DIV id="div1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:normal;">
      </DIV>
      <P>Click the section below to edit it's contents.</P>
      <DIV id="div2" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:normal;">
      </DIV>
      <P>Click the section below to edit it's contents.</P>
      <DIV id="div3" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:normal;">
      </DIV>
      <P>Click the following button to display the content for the editable section: <INPUT type="button"
        name="divButton"
        value="Display Editable Sections"
        onClick="displayEditableSectionDIVIDs();" >
      </P>
    </FORM>
    <SCRIPT language="JavaScript">
      var editlive_js;
      editlive_js = new EditLiveJava("editlive", "100%", "100%");
      editlive_js.setConfigurationFile("../redistributables/editlivejava/sample_eljconfig.xml");
      editlive_js.addEditableSection('div1');
      editlive_js.addEditableSection('div2');
      editlive_js.addEditableSection('div3');
    </SCRIPT>
  </BODY>
</HTML>
```

## See Also

- [Using Inline Editing Tutorial](#)

- [addEditableSection Method](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [setEditableSectionCSS Method](#)
- [setDisplayEditableMarker Method](#)

# getSelectedText Method

This method retrieves the currently selected text within the EditLive! applet. The method will retrieve the currently selected text and any inline tags within the current selection; it will not retrieve the parent tags of the selected text.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.getSelectedText([jsFunct]);
```

## Parameters

**jsFunct**

This parameter is optional.

The JavaScript function which receives the retrieved selected text.

## Returns

The selected text.

## Example Synchronous (EditLive 8.1+)

Calling **getSelectedText** synchronously returns the currently selected text (including inline tags) as a string. This example retrieves the currently selected text and then saves the retrieved contents to a <textarea> with the name *selectedText*. The name of the EditLive applet is *editlive\_js*.

```
document.exampleForm.selectedText.value = editlive_js.getSelectedText();
```

## Example Asynchronous (EditLive 8.0+)

When calling **getSelectedText** asynchronously you must provide a JavaScript function as the callback parameter for the **getSelectedText** method. In this example the *retrieveSelectedText* callback function receives the currently selected text (including inline tags) as a string. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getSelectedText('retrieveSelectedText');
```

The JavaScript function specified as the callback parameter receives the contents of the editor and then saves the retrieved contents to a <textarea> with the name *selectedText*.

```
function retrieveSelectedText(src){
    document.exampleForm.selectedText.value = src;
}
```

## Remarks

This method is designed for use when selecting text within a single parent block tag. Using the method outside of this context may result in unexpected behaviour.

If using this method with selections which span multiple block tags, the opening parent tag of the block at the start of the selection and the closing parent tag of the block at the end of the selection will both be stripped from the retrieved content. For example:

If the following markup existed inside an instance of EditLive!:

```
<p>This is the START first paragraph</p>
<p>This paragraph is the second paragraph</p>
<p>This is the <b>third</b> paragraph and END contains some bold text</p>
```

And the selection extended from the word START in the first paragraph to the word END in the last paragraph, the following content would be returned:

```
START first paragraph</p>  
<p>This paragraph is the second paragraph</p>  
<p>This is the <b>third</b> paragraph and END contains some bold text
```

# getStyles Method

This method retrieves the styling information from the EditLive! applet. It will retrieve all style information which lies between the <STYLE> tags in the document's <HEAD>.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.getStyles([strJSFunc]);
```

## Parameters

### jsFunc

This parameter is optional.

The JavaScript function which receives the retrieved EditLive! style information.

## Returns

The EditLive! style information.

## Example Synchronous (EditLive 8.1+)

Calling **getStyles** synchronously returns the style information from the editor as a string. This example retrieves the style information and then saves the style information to a <textarea> with the name *styleInfo*. The name of the EditLive applet is *editlive\_js*.

```
document.exampleForm.styleInfo.value = editlive_js.getStyles();
```

## Example Asynchronous (EditLive 8.0+)

When calling **getStyles** asynchronously you must provide a JavaScript function as the callback parameter for the **getStyles** method. In this example the *retrieveStyles* callback function receives the style information from the editor as a string. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getStyles('retrieveStyles');
```

The *retrieveStyles* function receives the style information from the editor and then saves the retrieved styles to a <textarea> with the name *styleInfo*.

```
function retrieveStyles(styles){
    document.exampleForm.styleInfo.value = styles;
}
```

## See Also

- [Retrieving Content From EditLive!](#)

# getWordCount Method

This method obtains a count of the number of words present within the EditLive! applet. It counts the number of words which lie between the <BODY> tags.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.getWordCount([ jsFunct ]);
```

## Parameters

### jsFunct

This parameter is optional.

The JavaScript function which receives the word count obtained from the EditLive! applet.

## Returns

The word count.

## Example Synchronous (EditLive 8.1+)

Calling **getWordCount** synchronously returns the the word count as a string. This example retrieves the word count and then displays it in a JavaScript alert dialog. The name of the EditLive applet is *editlive\_js*.

```
var wordCount = editlive_js.getWordCount();
```

## Example Asynchronous (EditLive 8.0+)

When calling **getWordCount** asynchronously you must provide a JavaScript function as the callback parameter for the **getWordCount** method. In this example the *wordCountAlert* callback function receives the word count as a string. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getWordCount('wordCountAlert');
```

The *wordCountAlert* function receives the word count as a string and displays it in a JavaScript alert dialog.

```
function wordCountAlert(count){
    alert('Word Count: ' + count);
}
```



# insertHTMLAtCursor Method

This method inserts developer-specified HTML at the cursor within the EditLive! applet. This method takes a JavaScript string as its only parameter.

## Syntax

JavaScript

```
editliveInstance.insertHTMLAtCursor(strHTML);
```

## Parameters

### strHTML

The string containing the HTML to be inserted at the cursor within the EditLive! applet.

## Example

The following code creates a <TEXTAREA>, named *htmlToInsert*, that will have its contents inserted into an instance of EditLive! at the cursor position via the **insertHTMLAtCursor** method. The **insertHTMLAtCursor** method will be associated with a HTML button. The name of the EditLive! applet is *editlivejs*.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript">
      </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>EditLive! contents will be loaded from here</P>
      <!--Create a textarea to load the applet contents from-->
      <P>
        <TEXTAREA name="htmlToInsert" cols="40" rows="10">
          <p>Content to be inserted</p>
        </TEXTAREA>
      </P>
      <P>Click this button to insert XHTML at the cursor in EditLive!</P>
      <P>
        <INPUT type="button"
          name="button1"
          value="Insert XHTML"
          onClick="editlivejs.insertHTMLAtCursor(encodeURIComponent(document.exampleForm.htmlToInsert.value ));">
      </P>
      <!--Create an instance of EditLive!-->
      <SCRIPT language="JavaScript">
        var editlivejs = new EditLiveJava("editlive", 450, 275);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(encodeURIComponent("<P>This is EditLive!</P>"));
        editlivejs.show();
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

The string passed to the JavaScript **insertHTMLAtCursor** method must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript `encodeURIComponent` function does not fully comply with the URL encoding standard.

## See Also

- [Retrieving Content From EditLive!](#)

# insertHyperlinkAtCursor Method

This method applies a developer-specified hyperlink to the text selected within EditLive!. If no text is selected, then the word in which the cursor is currently located will be selected and a hyperlink applied to it (this is the same functionality as the Insert Hyperlink button in EditLive!). The first argument of this method must be the URL corresponding to the hyperlink. This argument is mandatory. This method can also be used with an optional list of XHTML attributes as arguments. These arguments are then used as attributes for the hyperlink (<A>) tag and therefore must be valid XHTML attribute-value pairs.

## Syntax

JavaScript

```
editliveInstance.insertHyperlinkAtCursor(strHyperlink, [strAttribute]*);
```

## Parameters

### strHyperlink

This is a required parameter.

This string contains the hyperlink to be inserted at the cursor within the EditLive! applet.

### strAttribute

This is an optional parameter.

A valid XHTML attribute-value pair for the <A> (hyperlink) XHTML element. Name-value pairs must appear as they would within the XHTML source (i.e. with correct use of quotation marks).

## Example

The following code creates a text <INPUT>, named *hyperlinkToInsert*, that will have its contents inserted into an instance of EditLive! at the cursor position via the **insertHyperlinkAtCursor** method. The **insertHyperlinkAtCursor** method will be associated with a HTML button. The name of the EditLive! applet is *editlivejs*.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript">
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>The selected hyperlink will be inserted into EditLive!</P>
      <!--Create a text input to load the applet contents from-->
      <P>
        <SELECT name="hyperlinkToInsert" size=3>
          <OPTION value="http://www.ephox.com" SELECTED> Ephox
          <OPTION value="http://www.ephox.com/product/editliveforxml/default.asp"> EditLive!
          <OPTION value="mailto:someone@yourserver.com">Email Link
        </SELECT>
      </P>
      <P>Click here to insert hyperlink at the cursor in EditLive!</P>
      <P>
        <INPUT type="button"
          name="button1"
          value="Insert Hyperlink"
          onClick="editlivejs.insertHyperlinkAtCursor(document.exampleForm.hyperlinkToInsert.value);">
      </P>
      <!--Create an instance of EditLive!-->
      <SCRIPT language="JavaScript">
        var editlivejs = new EditLiveJava("editlive",450 , 275);
        editlivejs.setConfigurationFile("sample_config.xml");
        editlivejs.setBody(encodeURIComponent("<p>Contents of EditLive! for Java</p>"));
        editlivejs.show();
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

The following example demonstrates how to use the **insertHyperlinkAtCursor** with multiple arguments used to specify the attributes of the hyperlink (<A> tag) to be inserted into EditLive!. This example demonstrates how the *target* (*frame1*) and *name* (*hyperlink1*) attributes can be specified for the hyperlink *http://www.ephox.com*. Note the use of double and single quotation marks in this example.

```
editlivejs.InsertHyperlinkAtCursor("http://www.ephox.com", "target='frame1'", "name='hyperlink1'");
```

## Remarks

Note that this method can be used with differing numbers of arguments. However, the first argument of the method must always be the URL for the hyperlink and is not optional.

Arguments which represent XHTML attribute-value pairs must be valid XHTML. This includes the usage of the correct quotations marks.

See Also

- [Retrieving Content From EditLive!](#)

# isDirty Method

This method returns a string depicting whether the current contents of EditLive! have changed since the content was initially loaded into the editor.

Prior to version 8.1 this function had no return value and the function was asynchronous. While asynchronous use is still supported, since version 8.1 this function can now be used synchronously with a return value.

## Syntax

JavaScript

```
String editliveInstance.isDirty([jsFunct]);
```

## Parameters

### jsFunct

This parameter is optional.

The JavaScript function that receives the string (*TRUE* or *FALSE*) depicting whether the contents of EditLive! have changed since first being initialized.

## Returns

The value *TRUE* if the contents of EditLive! have changed since first being initialized.

## Example Synchronous (EditLive 8.1+)

Calling **isDirty** synchronously returns a string depicting whether the current contents of the editor have changed. This example retrieves the dirty status of the editor and then displays it in a JavaScript alert dialog. The name of the EditLive applet is *editlive\_js*.

```
var editorIsDirty = editlive_js.isDirty();
```

## Example Asynchronous (EditLive 8.0+)

When calling **isDirty** asynchronously you must provide a JavaScript function as the callback parameter for the **isDirty** method. In this example the *dirtyStatus* function receives the dirty status of the editor. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.isDirty('dirtyStatus');
```

The *dirtyStatus* function receives the dirty status of the editor and displays a JavaScript alert dialog based on the received status.

```
function dirtyStatus(status) {
    if(status == "true"){
        alert("Content Has Changed");
    }
    else {
        alert("Content Has NOT Changed");
    }
}
```

# openEditableSection Method

This method opens a specified Inline Editing section for editing, loading the editor applet in place of the DIV. The user can also click on the section to open it.

For more information, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Using Inline Editing](#) tutorial.

## Syntax

JavaScript

```
editliveInstance.openEditableSection(divID);
```

## Parameters

**divID**

The ID attribute for a HTML DIV to open. This DIV must be registered as an Inline Editing Section. For more information, see the [addEditableSection Method](#).

## Examples

The following code registers an Inline Editing section and provides a button to open it.

JavaScript

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="dashedDiv1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:
normal;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("dashedDiv1");
</script>

<input type="button" value="Open Section" onclick="editlivejava.openEditableSection('dashedDiv1');" />
```

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [setEditableSectionCSS Method](#)
- [setDisplayEditableMarker Method](#)
- [closeActiveEditableSection Method](#)
- [removeEditableSection Method](#)

# postDocument Method

This method allowed developers to make EditLive! use HTTP POST to submit its content directly to a POST acceptor script. EditLive! would also process the HTTP response. The response was processed in accordance with a parameter passed to the JavaScript method.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will perform no operation.

## See Also

- [Retrieving Content From EditLive!](#)

# removeEditableSection Method

This method will revert a specified Inline Editing DIV back to a standard HTML DIV. After performing this method on a specified DIV, clicking on the DIV will no longer display EditLive!.

If the specified DIV currently contains EditLive!, calling this method will remove the applet from the DIV and replace the DIV's content with the content in EditLive!.

For more information, see the [Using Inline Editing](#) article in the [Developer Guide](#) for this SDK, or undertake the [Using Inline Editing Tutorial](#).

## Syntax

JavaScript

```
editliveInstance.removeEditableSection(divID);
```

## Parameters

**divID**

The ID attribute for an Inline Editing DIV to unassociate from EditLive!. This DIV must be registered as an Inline Editing Section. For more information see the [addEditableSection Method](#).

## Example

The following code features a page with a DIV registered as an Inline Editing section. Pressing the **Remove Editor** button will revert the Inline Editing DIV back to a normal HTML DIV unassociated with EditLive!.

JavaScript

```
<script language="Javascript" src="../../redistributables/editlivejava/editlivejava.js"/>
<div id="dashedDiv1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:
normal;">
</div>
<script language="Javascript">
    editlivejs = new EditLiveJava("ELApplet", "100%", "100%");
    editlivejs.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.xml");
    editlivejs.addEditableSection("dashedDiv1");
</script>

<input type="button" value="Remove Editor" onclick="editlivejs.removeEditableSection('dashedDiv1');" />
```

## See Also

- [Using Inline Editing Tutorial](#)
- [getContentForEditableSection Method](#)
- [setContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [setEditableSectionCSS Method](#)
- [setDisplayEditableMarker Method](#)
- [openEditableSection Method](#)
- [closeActiveEditableSection Method](#)

# setBackgroundMode Method

This function hides the EditLive! applet and replaces it with a placeholder so that DOM elements can display on top of the applet. This overcomes a z-order bug in Java and is useful for AJAX applications.

In most browsers, the placeholder is an actual screenshot of the applet, so the process is smooth and the user will most likely not notice the transition. In Internet Explorer 6 and 7, the placeholder is just a grey box due to browser limitations.

## Syntax

### JavaScript

```
editliveInstance.setBackgroundMode(mode, callback);
```

## Parameters

### mode

A boolean that specifies the mode to change to.

- *true* changes to background mode, hiding the applet and replacing it with a placeholder.
- *false* changes back to normal mode.

### callback

Reference to a function that is invoked when the mode change completes. This reference can either be a javascript function reference, or the fully-qualified name of a javascript function.

You should wait for the callback to fire before performing other actions (e.g. displaying AJAX lightboxes)

8.0.1 and below: The callback is only fired if the mode actually changes, e.g. if you set background mode to *true* and it's already *true*, then the callback is not called.

8.0.2 and above: The callback is always fired.

9.0.1 and below: This function requires the **setBackgroundMode plugin**. To enable this, ensure the following line is in your config, under <plugins>

```
<plugin name="setBackgroundMode" />
```

9.0.2 and above: The plugin tag for setBackgroundMode is not required to enable this capability. If you leave this tag in your configuration file you will see errors in the EditLive! debug log, but EditLive! will function properly.

9.1.0.185 and above: A click on the placeholder image will deactivate background mode (no callback is fired). This is a safety measure. If for some reason the editor is still in background mode after hiding all DOM elements that covered the editor, the editor can now be "reactivated" by clicking on it.



# setBody Method (Run Time)

This method sets the contents of the EditLive! applet between the <BODY> tags. It will replace any existing contents of the applet with the contents the function is provided with as its parameter. This method takes a JavaScript string as its only parameter.

## Syntax

JavaScript

```
editliveInstance.setBody(strBody);
```

## Parameters

### strBody

The string representing the contents to be placed into the EditLive! applet between the <BODY> tags.

## Examples

The following code creates a <TEXTAREA>, named *bodyContents*, that will have its contents loaded into an instance of EditLive! via the **setBody** method. The **setBody** method will be associated with a HTML button. The name of the EditLive! applet is *editlive\_js*.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript">
      </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>EditLive! contents will be loaded from here</P>
      <!--Create a textarea to load the applet contents from-->
      <P>
        <TEXTAREA name="bodyContents" cols="40" rows="10">
          <p>Content to be loaded</p>
        </TEXTAREA>
      </P>
      <P>Click this button to set applet contents</P>
      <P>
        <INPUT type="button"
          name="button1"
          value="Set Contents"
          onClick="editlive_js.setBody(encodeURIComponent(document.exampleForm.bodyContents.value));">
      </P>
      <!--Create an instance of EditLive!-->
      <SCRIPT language="JavaScript">
        var editlive_js = new EditLiveJava("editlive", 450, 275);
        editlive_js.setConfigurationFile("sample_elconfig.xml");
        editlive_js.show();
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

The string passed to the JavaScript **setBody** method must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript *encodeURIComponent* function does not fully comply with the URL encoding standard.

## See Also

- [Retrieving Content From EditLive!](#)

# setContentForEditableSection Method

This method loads HTML content into a specified Inline Editing section (represented either by a DIV or an instance of EditLive!).

For more information on the Inline Editing functionality for EditLive!, see the [Using Inline Editing](#) article.

## Syntax

JavaScript

```
editliveInstance.setContentForEditableSection(divID, html);
```

## Parameters

### divID

This parameter is required.

The ID attribute for the DIV registered as an inline editing section with EditLive! (via the [addEditableSection Method](#)).

### html

This parameter is required.

The HTML to be inserted into the inline editing section (represented by either a DIV or an instance of EditLive!). This HTML can be either a HTML fragment or an entire HTML document.

The value of the html parameter must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript `encodeURIComponent` function does not fully comply with the URL encoding standard.

## Example

The following code creates a webpage featuring a DIV. This DIV is registered with EditLive! as an inline editing section. The *Load Content* button will load the text **New Content** into the inline editing section.

If the user clicks the DIV, EditLive! will load in its place. Clicking the *Load Content* button would then load the string **New Content** into the editor instance.

If the user does not click the DIV, pressing *Load Content* will load to the string **New Content** into the DIV itself.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"/>
    <SCRIPT language="JavaScript">
      function loadEditableSectionContent(){
        editlive_js.setContentForEditableSection('div1', encodeURIComponent('<p><b>New Content</b></p>'));
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>Click the section below to edit it's contents.</P>
      <DIV id="div1" style="width: 600px; height: 400px; overflow: auto; border: 1px dotted gray; white-space:normal;">
      </DIV>
      <P>Click the following button to load the text <b>New Content</b> into the editable section: <INPUT
type="button"
      name="divButton"
      value="Load Content"
      onClick="loadEditableSectionContent();" >
    </P>
  </FORM>
  <SCRIPT language="JavaScript">
    var editlive_js = new EditLiveJava("editlive", "100%", "100%");
    editlive_js.setConfigurationFile("../redistributables/editlivejava/sample_eljconfig.xml");
    editlive_js.addEditableSection('div1');
```

```
</SCRIPT>  
</BODY>  
</HTML>
```

## See Also

- [Using Inline Editing Tutorial](#)
- [addEditableSection Method](#)
- [getContentForEditableSection Method](#)
- [getEditableSections Method](#)
- [setEditableSectionCSS Method](#)
- [setDisplayEditableMarker Method](#)

# setDocument Method (Run Time)

This method sets the contents of the EditLive! applet between the <HTML> tags. It will replace any existing contents of the applet with the contents the method is provided with as its parameter. This method takes a JavaScript string as its only parameter.

## Syntax

JavaScript

```
editliveInstance.setDocument(strDocument);
```

## Parameters

### strDocument

A string of the contents to be placed into the EditLive! applet.

## Example

The following code creates a <TEXTAREA>, named *documentContents*, that will have its contents loaded into an instance of EditLive! via the **setDocument** method. The **setDocument** method will be associated with a HTML button. The name of the EditLive! applet is *editlivejs*.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript">
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>EditLive! contents will be loaded from here</P>
      <!--Create a textarea to load the applet contents from-->
      <P>
        <TEXTAREA name="documentContents" cols="40" rows="10">
          <html><body><p>Content to be loaded</p></body></html>
        </TEXTAREA>
      </P>
      <P>Click this button to set applet contents</P>
      <P>
        <INPUT type="button"
          name="button1"
          value="Set Contents"
          onClick="editlivejs.setDocument(encodeURIComponent(document.exampleForm.documentContents.value));">
      </P>
      <!--Create an instance of EditLive!-->
      <SCRIPT language="JavaScript">
        var editlivejs;
        editlivejs = new EditLiveJava("editlive", 450, 275);
        editlivejs.setConfigurationFile("sample_config.xml");
        editlivejs.show();
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

The string passed to the JavaScript **setDocument** method must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript `encodeURIComponent` function does not fully comply with the URL encoding standard.

## See Also

- [Retrieving Content From EditLive!](#)

# setProperty Method

This method, when given a string of relevant name-value pairs, sets the attributes for an instance of a specific tag within EditLive!. This method is designed to be used with custom menu and/or toolbar items that call functions passing tag attributes. For more information on how to use custom menu and toolbar items with EditLive!, please see the [Creating Custom Menu and Toolbar Items](#) article.

## Syntax

JavaScript

```
editliveInstance.setProperty(strProperties);
```

## Parameters

### strProperties

This string provides a list of name-value pairings of attributes for the relevant HTML tag. Each name and value for each pairing must be delimited by an equal sign (=) character. Name-value pairings must be delimited by a new line (\n) character.

## Example

The following sets the properties for an instance of a tag inside an instance of EditLive! named *editlivejs*. The function *setNewProperties* (called by a custom menu or toolbar item) will display the name-value pairs of the tag currently selected.

```
//set up an instance of EditLive!  
var editlivejs = new EditLiveJava("editlive", 700, 400);  
editlivejs.setConfigurationFile("sample_config.xml");  
editlivejs.show();  
  
...  
  
//create a function which sets properties  
function setNewProperties(newProperties){  
    editlivejs.setProperty(newProperties);  
}
```

The string passed to the JavaScript **setProperty** property must be [URL encoded](#). It is recommended that a server-side URL encoding function be used if available as the JavaScript encodeURI function does not fully comply with the URL encoding standard.

## Remarks

Each name and value for each pairing must be delimited by an equal sign (=) character.

Name-value pairings must be delimited by a new line (\n) character.

In order to correctly set the properties of the relevant tag, it should be ensured that the *ephoxTagID* attribute is not altered by the functions external to EditLive!. Also, the tag attribute must be present and the value of this attribute must correspond to the name of the tag (i.e. *span* for a *<span>* tag).

The value of the attribute with the name of *tag* designates the type of tag for which the properties are relevant. Changing the value of the tag attribute will change the tag type in EditLive!. Thus, if the value of a tag attribute with the value *td* was changed to *th*, then the relevant table cell would be changed from a normal (*td*) cell to a table header (*th*) cell.

The tag selected may contain standalone attributes. These are attributes which have only a name and do not exist as a name-value pairing (for example, the *NOWRAP* attribute of the *<td>* tag). In order to add such an attribute to the properties string, a name-value pair in which the name and value are the same (e.g. *NOWRAP=NOWRAP*) should be added to the properties string.

## See Also

- [Creating Custom Menu and Toolbar Items](#)

# uploadImages Method

This method uploads any local images within the instance of EditLive! to the Web server. Upon calling this method the URLs for local images within the applet will be changed to point to the copies on the Web server instead of the local copies.

Prior to version 8.0 this function was asynchronous. The callback notification mechanism is still supported, however since version 8.0 this function can now be used synchronously.

## Syntax

### JavaScript

```
editliveInstance.uploadImages();
```

## Parameters

### jsFunct

This parameter is optional.

The JavaScript function which is called after all local images have been uploaded.

## Example

The following code allows the user to force an upload of the images in EditLive! to the Web server. The **uploadImages** method is called by clicking on a button within the form.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! JavaScript Example</TITLE>
    <!--Include the EditLive! JavaScript Library-->
    <SCRIPT src="editlivejava/editlivejava.js" language="JavaScript"></SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>
        <INPUT type="button"
          name="button1"
          value="Upload Images"
          onClick="editlivejs.uploadImages();" >
      </P>
      <!--Create an instance of EditLive!-->
      <SCRIPT language="JavaScript">
        var editlivejs = new EditLiveJava("editlive",450 , 275);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(encodeURIComponent("<p>Some initial text</p>"));
        editlivejs.show();
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

## Remarks

The **uploadImages** method should be called prior to using the [GetBody](#) or [GetDocument](#) load-time properties to retrieve the contents of the EditLive! applet.

The **uploadImages** method need not be called if the contents of EditLive! are processed through a server-side script using a HTTP Post. For more information on using HTTP Posts to retrieve EditLive! content, see the [Retrieving Content From EditLive!](#) article.

The upload script used by this method is the script specified in the EditLive! configuration file through the [<httpUploadData>](#) element.

## See Also

- [Retrieving Content From EditLive!](#)
- [<httpUploadData>](#) Configuration File Element

# getContent Method

This function retrieves the contents of the EditLive! applet, **according to the [returnBodyOnly](#) setting**.

This method is new in EditLive! 8.1.

## Syntax

JavaScript

```
String editliveInstance.getContent([jsFunct],[blnUploadImages]);
```

## Parameters

### jsFunct

This parameter is optional.

The JavaScript function which receives the retrieved EditLive! applet contents.

### blnUploadImages

This parameter is optional.

This is a boolean which indicates whether images should be uploaded to the server when this function is called. The uploading of images will occur immediately before the content is retrieved.

The default value is *false*.

## Returns

The contents of the EditLive! editor. This will either be a full HTML document, or the contents within the <body> tag, according to the [returnBodyOnly](#) setting.

## Synchronous Example (EditLive 8.1+)

Calling **getContent** synchronously returns the appropriate contents of the editor as a string. This example retrieves the contents of the editor and then saves the retrieved contents to a <textarea> with the ID of *contents*. Images will be uploaded to the server when **getContent** is called as *blnUploadImages* is set to true. The name of the EditLive applet is *editlive\_js*.

```
document.getElementById('contents').value = editlivejs.getContent();
```

## Asynchronous Example (EditLive 8.1+)

When calling **getContent** asynchronously you must provide a JavaScript function as the callback parameter for the **getContent** method. In this example the *retrieveContent* callback function receives the appropriate contents of the editor as a string. Images will be uploaded to the server when **getContent** is called as *blnUploadImages* is set to *true*. The name of the EditLive applet is *editlive\_js*.

```
editlive_js.getContent("retrieveContent", true);
```

The *retrieveContent* function receives the contents of the editor and then saves the retrieved contents to a <textarea> with the name *bodyContents*.

```
function retrieveContent(src){
    document.exampleForm.bodyContent.value = src;
}
```

## Remarks

When uploading locally stored images to a web server for an instance of EditLive!, ensure that the *blnUploadImages* parameter is set to *true* when calling this method.

## See Also

- [getBody Method](#)

- [getDocument Method](#)
- [getContentForEditableSection Method](#)
- [setReturnBodyOnly Method](#)



# performRaiseEvent method

This methods allows users of the advanced API to invoke Java code in their custom plugin from Javascript. A [TextEvent](#) of Raise Event action will be triggered when this method is called with the eventName argument as the value of the extraString on the TextEvent.

Available since EditLive 8.1.0.124

It is important to call `Event.setHandled(true)` when the event has been handled. If `setHandled` is not set to true EditLive will attempt to invoke a Javascript function using the value of the `eventName` parameter.

## Syntax

### JavaScript

```
editliveInstance.performRaiseEvent(eventName);
```

## Parameters

### eventName

This is a string representing the name of the event to be called in the custom Java plugin.

## Example

Calling `performRaiseEvent` will trigger a Java `TextEvent`. In this case a Raise Event Action of the value of "doSomething" will be raised.

```
editlive_js.performRaiseEvent("doSomething");
```

In the custom Java plugin this `TextEvent` must be caught. It is very important that the `setHandled(true)` is called on the event otherwise EditLive will fallback to calling a javascript function with the name of `eventName` value. [The Creating Plugins Utilizing Advanced APIs Tutorial](#) has an example on how to write a custom plugin and listen to `TextEvents`.

```
import javax.swing.JOptionPane;
import com.ephox.editlive.ELJBean;
import com.ephox.editlive.common.EventListener;
import com.ephox.editlive.common.TextEvent;
import com.ephox.editlive.plugins.Plugin;
import com.ephox.editlive.plugins.SafeLoadingPlugin;

public class AdvancedAPIPlugin extends SafeLoadingPlugin implements EventListener, Plugin {

    public AdvancedAPIPlugin(final ELJBean bean) {
        super(bean, "8", "insertHTML");
    }

    @Override
    public void raiseEvent(TextEvent e) {
        if (e.isRaiseEventAction("doSomething")) {
            e.setHandled(true);
            JOptionPane.showMessageDialog(null, "Did something");
        }
    }

    @Override
    protected void editorInitialized() {
        getBean().getEventBroadcaster().registerBeanEditorListener(this);
    }
}
```

# isEditableSectionDirty Method

This method returns a string depicting whether the current contents of an editable section have changed since the content was initially loaded into the editor.

Info



This method was introduced in EditLive 9.0.3.65

## Syntax

JavaScript

```
boolean editliveInstance.isEditableSectionDirty(sectionId);
```

## Parameters

**sectionId**

String. ID of the section to get the dirty status for. This is the same value used to create the section with the `addEditableSection` method.

## Returns

*true* if the contents of the section have changed since first being initialized.

Throws an error if the section is not found.

# setIsDirty Method

This method can be used to indicate that the current content of EditLive! has changed since the content was initially loaded into the editor.

This function was introduced in EditLive! 9.0.3.146.

## Syntax

JavaScript

```
editliveInstance.setIsDirty(dirty);
```

## Parameters

**dirty**

A boolean that specifies if the content of EditLive! has changed since the content was initially loaded into the editor.

## Example (EditLive 9.0.3.146+)

```
editlive_js.setIsDirty(true);
```

# Utility Methods

The utility functions for EditLive! are Javascript functions designed to help developers integrate the editor into their system. The utility functions do not directly communicate with EditLive!; rather, these functions help to optimize the environment where the editor will be operating.

# quickStart Method

Depending on the user's web browser, this function performed several operations to optimize the loading time for the editor.

- For Mozilla browsers, such as Firefox and the Mozilla Suite, this function would preload the Java Virtual Machine (JVM).
- For Internet Explorer, this function would preload the JVM and download the core EditLive! files in order to optimize the loading time for the editor.
- For all other browsers, this function had no effect.

Removed in EditLive! 8.0



This function was removed in EditLive! 8.0. If called, it will fail with an exception.

## See Also

- [Optimizing Load Time](#)
- [Optimizing Load Times Tutorial](#)

# Configuration File Elements

An EditLive! configuration file uses XML to configure and customize the behavior and functionality of EditLive!. Functionality may be turned on or off, aspects of behaviour changed or custom commands introduced.

Because server-side languages can be used to generate documents of any types at run-time, configuration files can also be stored as server-side language files (e.g. JSP files, ASP files) as long as the file contains calls to render the information as XML at run-time.

The following load-time properties are used to specify the EditLive! configuration file in the applet:

- [setConfigurationFile Method](#)
- [setConfigurationText Method](#)

The following methods of the JavaBean are used to specify the configuration file in the Swing SDK:

- [setConfigurationDOM](#)
- [setConfigurationFile](#)
- [setConfigurationText](#)
- [setConfigurationURL](#)

## Configuration File Elements Grouped by Functionality

### Editor Behaviour and Settings

[ephoxLicenses](#)  
[license](#)  
[otherLicenses](#)  
[webeqLicense](#)  
[excellImport](#)  
[htmlImport](#)  
[textImport](#)  
[wordImport](#)  
[wysiwygEditor](#)  
[sourceEditor](#)  
[htmlFilter](#)  
[customTags](#)  
[spellCheck](#)  
[thesaurus](#)  
  
[trackChanges](#)  
  
[doubleClickActions](#)  
  
[action](#)  
  
[colorPalette](#)  
  
[color](#)  
  
[document](#)  
[html](#)  
[head](#)  
[base](#)  
[link](#)  
[meta](#)  
[style](#)  
[title](#)  
[body](#)  
[httpUpload](#)  
[httpUploadData](#)  
[authentication](#)  
[realm](#)  
[webdav](#)  
[repository](#)

### Miscellaneous

[editLive](#)  
[plugins \(config\)](#)

### Toolbar and Menu Settings

[menuBar](#)  
[menu](#)  
[menuItem](#)  
[menuItemGroup](#)  
[submenu](#)  
[customMenuItem](#)  
[menuSeparator](#)  
[shortcutMenu](#)  
[shrtMenu](#)  
[shrtMenuItem](#)  
[shrtMenuSeparator](#)  
[toolbars](#)  
[toolbar](#)  
[toolbarButton](#)  
[toolbarButtonGroup](#)  
[toolbarComboBox](#)  
[comboBoxItem](#)  
[customToolBarButton](#)  
[customToolBarComboBox](#)  
[customComboBoxItem](#)  
[toolbarSeparator](#)

### Media Settings

[mediaSettings](#)  
[httpUpload](#)  
[hyperlinks](#)  
[images](#)  
[multimedia](#)  
[mathml](#)  
[multimedia](#)  
[services](#)  
[service](#)  
[types](#)  
[type](#)  
[param](#)  
[webdav](#)  
[repository](#)

### Dialog Settings

[hyperlinks](#)  
[hyperlinkList](#)  
[hyperlink](#)  
[mailtoList](#)  
[mailtoLink](#)  
  
[placesInDocumentList](#)  
  
[webdav](#)  
  
[repository](#)  
[images](#)  
[imageList](#)  
  
[image](#)  
  
[imageBrowser](#)  
  
[imageDialog](#)  
  
[webdav](#)  
  
[repository](#)  
[templates](#)  
[category](#)  
[template](#)  
[accessibilityChecks](#)  
[colorPalette](#)  
[color](#)  
[symbols](#)  
[symbol](#)  
[contentLanguages](#)  
[language](#)

# accessibilityChecks

This element allows developers to customize the EditLive! [Accessibility Checker](#) dialog.

## Configuration Element Tree Structure

`<editLive>`

`<accessibilityChecks>`

```
<editLive>
  ...
  <accessibilityChecks ... />
  ...
</editLive>
```

## Optional Attributes

### errors

This option specifies whether any errors against the selected Accessibility Guidelines are displayed. Errors depict HTML elements which require a precise change to ensure the content is accessible under the selected guidelines.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *true*

### warnings

This option specifies whether any warnings against the selected Accessibility Guidelines are displayed. Warnings related to the entire HTML document itself, where errors focus on a particular HTML element.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *true*

### manual checks

This option specifies whether to do display aesthetic based accessibility checks that can't be programmatically detected. As a user, you will need to review your content to ensure these accessibility guidelines are met.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *false*

### WCAG2A

This option specifies whether errors, warnings, or manual checks will be displayed based on the W3C (World Wide Web Consortium) Web Content Accessibility Guideline version 2.0. This guideline specifies that a web content developer must specify these checkpoints.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *true*

### WCAG2AA

This option specifies whether errors, warnings or manual checks will be displayed based on the W3C (World Wide Web Consortium) Web Content Accessibility Guideline version 2.0. This guideline specifies that a web content developer should specify these checkpoints.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *true*

## Section 508

This option specifies whether errors, warnings, or manual checks will be displayed based on the Section 508 guidelines. Section 508 are web content accessibility guidelines specified as part of the US Rehabilitation Act.

This attribute has three possible values:

- *true* - The option will first appear checked,
- *false* - The option will first appear unchecked, or
- *hidden* - The option will not appear at all for users.

Default Value: *true*

## emptyImageAlt

This option specifies whether empty alt attributes on images result in errors, warnings, or are ignored.

This attribute has three possible values:

- *none* - Ignore empty alt attributes,
- *warn* - Produce a warning in the accessibility checker and inline accessibility, or
- *error* - Produce an error in the accessibility checker and inline accessibility.

Default Value: *error*

## tableMappingIssues

For tables containing cells that aren't either headers or data cells mapped to headers, this configuration option allows developers to specify whether the table is flagged as an accessibility error or warning.

- *warn* - Produce a warning in the accessibility checker and inline accessibility, or
- *error* - Produce an error in the accessibility checker and inline accessibility.

Default Value: *error*

## inlineAccessibility

This configuration option allows you to enable or disable the [Accessibility As You Type](#) functionality by default.

This attribute is a boolean and can only be *true* or *false*.

Default Value: *false*

## Example

The following example demonstrates how to set the various attributes of the `<accessibilityChecks>` element.

```
<editLive>
  ...
  <accessibilityChecks
    errors="true"
    warnings="false"
    manual="hidden"
    WCAG2A="true"
    WCAG2AA="true"
    Section508="hidden" />
  ...
</editLive>
```

## Remarks



If the configuration file specified for EditLive! does not contain an **<accessibilityChecks>** element, any changes a user makes to the Accessibility Checker dialog will be saved on that user's machine.

### Example

A user checks the *Errors*, *Warnings*, and *Section 508* checkboxes in the Accessibility Checker dialog. They leave all other checkboxes unchecked. The next time the user displays this dialog, regardless of whether EditLive! is rendered in a completely different webpage, the *Errors*, *Warnings*, and *Section 508* checkboxes will be checked.

**action**

# action (Applet)

This element specifies a custom action to be performed with empty tags (i.e. tags with no body such as <img>) within EditLive!.

The action element only supports standard HTML tags. The action element cannot be used to target custom tags.

## Configuration Element Tree Structure

```
<editLive>
<wysiwygEditor>
<customTags>
<doubleClickActions>
<action (Applet)>
```

```
<editLive>
  ...
  <wysiwygEditor>
    <customTags>
      <doubleClickActions>
        <action ... />
      </doubleClickActions>
    </customTags>
  </wysiwygEditor>
  ...
</editLive>
```

## Required Attributes

### name

The empty tag for which this action applies.

### action

The action which this custom action performs. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string.
- *PostDocument* - Post the content of the applet to a server-side script.

The actions available for empty tags are the same actions available for custom menu and toolbar items. For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the *action* attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string ##ephox##. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! uses to POST its content.  
This parameter is required.
  - *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java is to POST to.  
The parameter is required.
  - *Response Processing*  
The operation that EditLive! is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
    1. *saveToDisk* - Presents the user with a save file dialog, with which they can save the response to the local machine.
    2. *callback* - Passes the entire content of the HTTP response to a specified JavaScript callback function for processing.  
This parameter is required.
  - *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.

This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the *value* attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3Cp%3E`.

## Examples

The following example demonstrates how to define a JavaScript method to be raised upon double clicking any `<img>` tag. The name of the JavaScript function to be raised is *jsFunction*.

```
<editLive>
...
<wysiwygEditor>
  <customTags>
    <doubleClickActions>
      <action
        name="img"
        action="raiseEvent"
        value="jsFunction"
      />
    </doubleClickActions>
  </customTags>
  ...
</wysiwygEditor>
...
</editLive>
```

For more examples of using the `<action>` configuration file element, see the Custom Operations for Menu and Toolbar Items section of the [Creating Custom Menu and Toolbar Items](#) article. The functionality and format of the `<customMenuItem>`, `<customToolbarButton>`, and [Creating Custom Menu and Toolbar Items](#) `<customComboBoxItem>` attributes is the same as the `<action>` attributes.

## Remarks

The `<action>` element can appear multiple times within the `<doubleClickActions>` element.

The `<action>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<action name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

The actions available for empty tags are the same actions available for custom tags. For more information on these actions see the Custom Operations for Menu and Toolbar Items section of the [Creating Custom Menu and Toolbar Items](#) article.

## See Also

- [Encoding Content for Use with EditLive!](#)
- [Creating Custom Menu and Toolbar Items](#)

# action (Swing SDK)

This element specifies a custom action to be performed with empty tags (i.e. tags with no body such as `<img>`) within EditLive! for Java Swing.

The `action` element only supports standard HTML tags. The action element cannot be used to target custom tags.

## Configuration Element Tree Structure

```
<editLive>
<wysiwygEditor>
<customTags>
<doubleClickActions>
<action>
```

```
<editLive>
  ...
  <wysiwygEditor>
    <customTags>
      <doubleClickActions>
        <action ... />
      </doubleClickActions>
    </customTags>
  </wysiwygEditor>
  ...
</editLive>
```

## Required Attributes

### name

The empty tag for which this action applies.

### action

The action which this custom action performs. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the [ELJBean](#) will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the [ELJBean](#) will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *PostDocument* - Post the content of the applet to a server side script.

The actions available for empty tags are the same actions available for custom menu and toolbar items. For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the *action* attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! uses to POST its content.

- This parameter is required.  
 • *Post Acceptor URL*  
 The URL for the POST acceptor that EditLive! for Java is to POST to.  
 The parameter is required.
- *Response Processing*  
 The operation that EditLive! is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
  1. *saveToDisk* - Presents the user with a save file dialog, with which they can save the response to the local machine.
  2. *callback* - Passes the entire content of the HTTP response to a specified JavaScript callback function for processing.
 This parameter is required.
- *JavaScript Callback Function*  
 The name of the JavaScript callback function to use for processing the response.  
 This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the *value* attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3Cp%3E`.

### Examples

The following example demonstrates how to define a JavaScript method to be raised upon double clicking any `<img>` tag. The name of the JavaScript function to be raised is *jsFunction*.

```
<editLive>
...
<wysiwygEditor>
  <customTags>
    <doubleClickActions>
      <action
        name="img"
        action="raiseEvent"
        value="jsFunction"
      />
    </doubleClickActions>
  </customTags>
  ...
</wysiwygEditor>
...
</editLive>
```

For more examples of using the `<action>` configuration file element, see the Custom Operations for Menu and Toolbar Items section of the [Creating Custom Menu and Toolbar Items](#) article. The functionality and format of the `<customMenuItem (Swing SDK)>`, `<customToolbarButton (Swing SDK)>`, and `<creating Custom Menu and Toolbar Items <customComboBoxItem>` attributes is the same as the `<action>` attributes.

### Remarks

The `<action>` element can appear multiple times within the `<doubleClickActions>` element.

The `<action>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<action name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

The actions available for empty tags are the same actions available for custom tags. For more information on these actions see the Custom Operations for Menu and Toolbar Items section of the [Creating Custom Menu and Toolbar Items](#) article.

## See Also

- [Creating Custom Menu and Toolbar Items](#)

# authentication

WebDAV support has been removed in EditLive! 9.1

This element contains the settings required for a user of EditLive! to authenticate themselves to a specific Web server realm. The element should contain the settings for each realm that the instance of EditLive! concerned with may access.

## Configuration Element Tree Structure

[<editLive>](#)  
[<authentication>](#)

```
<editLive>
  <authentication>
    <!--authentication configuration settings-->
  </authentication>
  ...
</editLive>
```

## Child Elements

[<realm>](#)

This configuration element contains the authentication settings for a specific realm.

## Remarks

The [<authentication>](#) element can appear only once within the [<editLive>](#) element.



# base

This element provides the information which is to be stored as attributes within the <BASE> tag between the <HEAD> tags of the Tiny EditLive! document. The element has only a HREF attribute.

The value which appears within the <base> element will appear within the actual EditLive! document between the <HEAD> tags in a <BASE> tag.

## Configuration Element Tree Structure

```
<editLive>
<document>
<html>
<head>
<base>
```

```
<editLive>
  <document>
    <html>
      <head>
        <base ... />
      </head>
    </html>
  </document>
  ...
</editLive>
```

## Optional Attributes

### href

This specifies the URL of a document whose server path is to be used as the base URL for all relative references in the document.

## Examples

The following example demonstrates how to specify the URL *http://www.yourserver.com/* as the value for the base URL.

```
<editLive>
  <document>
    <html>
      <head>
        <base href="http://www.yourserver.com/" />
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Remarks

If the <base> element is not specified then EditLive! will use the base location of the page which it resides in as the base URL. For example, if the instance of EditLive! appeared in the page *http://www.yourserver.com/editlive.html*, then the instance of EditLive! in that page will use *http://www.yourserver.com* as its base URL.

The <base> element can appear only once within the <head> element.

The <base> element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<base href=.../>
```

## See Also

- [setBaseURL Method \(Applet\)](#)
- [setBaseURL\(\) method in ELJBean class \(Swing SDK\)](#)

# body

This element allows for the configuration of information which is contained as attributes within the <BODY> tag of the Tiny EditLive! document. The settings which are configured through the <body> element will appear in the actual document inside EditLive! as attributes within the <BODY> tag.

## Configuration Element Tree Structure

```
<editLive>
<document>
<html>
<body>
```

```
<editLive>
  <document>
    <html>
      ...
      <body ... />
    </html>
  </document>
  ...
</editLive>
```

## Optional Attributes

### alink

This attribute specifies the value for the **alink** attribute of the <BODY> tag within the actual EditLive! document. The alink attribute specifies the color of an active hyperlink (ie. a hyperlink being clicking on) within the document.

### background

This attribute specifies the value for the **background** attribute of the <BODY> tag within the actual EditLive! document. The background attribute specifies a background image for the body of the document.

### bgColor

This attribute specifies the value for the **bgColor** attribute of the <BODY> tag within the actual EditLive! document. The bgColor attribute specifies the background color of the entire document.

### link

This attribute specifies the value for the **link** attribute of the <BODY> tag within the actual EditLive! document. The link attribute specifies the color of a hyperlink which has not been visited within the document.

### text

This attribute specifies the value for the **text** attribute of the <BODY> tag within the actual EditLive! document. The text attribute specifies the color of text within the document.

### vlink

This attribute specifies the value for the **vlink** attribute of the <BODY> tag within the actual EditLive! document. The vlink attribute specifies the color of a hyperlink which has been visited within the document.

### contenteditable

This attribute specifies whether the contents of EditLive! will be read-only for the user of the editor.

This attribute is a boolean and can have the value of either *true* or *false*.

Default Value: *false*

## Example

This example demonstrates how to set various attributes for use in the <BODY> tag within EditLive! documents.

```
<editLive>
  <document>
    <html>
```

```
...
<body
  alink="#FF0000"
  background="watermark.jpg"
  bgColor="blue"
  link="#00FF00"
  text="red"
  vlink="#teal"
/>
</html>
</document>
...
</editLive>
```

## Remarks

The **<body>** element can appear only once within the **<html>** element. The **<body>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<body alink=... />
```

The body tag is only used to specify the attributes contained in the **<body>** tag of the document appearing in the EditLive! editor. Any HTML text written inside this xml tag will not appear in the document stored in the EditLive! editor.

# category

EditLive!'s [Template Browser](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

This element allows you to specify categories for the templates displayed in EditLive!'s Template Browser.

## Configuration Element Tree Structure

```
<editLive>  
<templates>  
<category>
```

```
<editLive>  
...  
<templates>  
  <category name="Category 1">  
    ...  
  </category>  
</templates>  
...  
</editLive>
```

## Required Attributes

### name

The name of the category. This will be displayed in the Template Browser.

## Child Elements

```
<category>
```

Any category can itself contain categories.

```
<template>
```

This element specifies a template that will appear in the Template Browser.

## Remarks

The **<category>** element can appear multiple times within the **<templates>** element.

# color

This element allows developers to specify a single color to be displayed in the color chooser used by EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<colorPalette>  
<color>
```

```
<editLive>  
  ...  
  <wysiwygEditor>  
    ...  
    <colorPalette>  
      <color />  
    </colorPalette>  
    ...  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Required Attributes

### name

This attribute defines a color to display in the color chooser. The value specified by this attribute needs to be the *hexadecimal* representation for the desired color (e.g. *name="#FF0000"*).

## Example

The following example depicts adding the colors *red*, *blue* and *yellow* to the color chooser used by EditLive!:

```
<editLive>  
  ...  
  <wysiwygEditor>  
    <colorPalette>  
      <color name="#FF0000" />  
      <color name="#0000FF" />  
      <color name="#FFFF00" />  
    </colorPalette>  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Remarks

The `<color>` element can appear any amount of times within the `<colorPalette>` element.

# colorPalette

This element allows you to customize the colors accessible via the color chooser dialog used by EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<colorPalette>
```

```
<editLive>  
  ...  
  <wysiwygEditor>  
    ...  
    <colorPalette>  
  
    </colorPalette>  
  <wysiwygEditor/>  
  ...  
</editLive>
```

## Optional Attributes

### showMoreColors

This boolean attribute defines whether the color chooser displayed by the editor will allow the user to select any color they wish (via the *More Colors* link on the dialog). If this value is set to *false*, the only colors available to the user will be the colors defined in the `<color>` child elements.

This attribute is a boolean and can have the value of either *true* or *false*.

Default Value: *true*

## Child Elements

```
<color>
```

These elements define a single color which will appear in the color chooser used by EditLive!.

## Example

The following example would disable the *More Colors* link from the color chooser (hence, limiting the user to only select colors specified by the `<color>` elements).

```
<editLive>  
  ...  
  <wysiwygEditor>  
    ...  
    <colorPalette showMoreColors="false">  
  
    </colorPalette>  
  <wysiwygEditor/>  
  ...  
</editLive>
```

## Remarks

The `<colorPalette>` element can appear only once within the `<editLive>` element.

# comboBoxItem

This element contains the information required by Tiny EditLive! to configure an item within one of the EditLive! combo boxes.

## Configuration Element Tree Structure

```
<editLive>  
<toolbars>  
<toolbar>  
<toolbarComboBox>  
<comboBoxItem>
```

```
<editLive>  
  ...  
  <toolbars>  
    <toolbar>  
      <toolbarComboBox ... >  
        <comboBoxItem ... />  
      </toolbarComboBox>  
    </toolbar>  
  </toolbars>  
  ...  
</editLive>
```

## Required Attributes

name

The value of the name attribute is different depending on the type of combo box being configured. The following gives information on the way that the name attribute is used in each case:

### Style Combo Box

This attribute gives the value to be used when the item is being inserted into the HTML source code. When used in the Style combo box the name attribute gives the name of the tag inserted into the HTML source code within EditLive!.

*Example:* If the name attribute was set to *H1* then the *<H1>* tag would be inserted into the HTML when this style was used.

### Typeface Combo Box

This attribute gives the value to be used when the item is being inserted into the HTML source code. When used in the Typeface combo box the name attribute gives the value used for the face attribute within the *<SPAN>* tag used within the EditLive! HTML source code.

*Example:* If the name attribute was set to Times New Roman then the following *<SPAN>* tag would be inserted into the EditLive! HTML source code:

```
<SPAN style=" font-family: 'Times New Roman';"></span>
```

### Size Combo Box

This attribute gives the value to be used when the item is being inserted into the HTML source code. When used in the Size combo box the name attribute gives the value used for the size attribute within the *<span>* tag used within the EditLive! HTML source code.

The value assigned to this attribute can be of the following formats:

#### W3C Accessible Sizing

Specifying a number between 1 and 7 will create a W3C Accessible value for the size attribute created. The following list details the W3C Accessible value created for each 1 - 7 value specified for the name attribute.

Name Attribute Value	Span Tag Created
1	<span style="font-size: xx-small"></span>
2	<span style="font-size: x-small"></span>
3	<span style="font-size: small"></span>
4	<span style="font-size: medium"></span>
5	<span style="font-size: large"></span>

6	<span style="font-size: x-large"></span>
7	<span style="font-size: xx-large"></span>

*Example:* The following combo-box item will display the text 12pt.

```
<comboBoxItem name="3" text="12pt" />
```

If the above combo-box item is selected, the following tag will be created:

```
<span style="font-size: small"></span>
```

### Size and Format Combination

You can specify the specific font size and format you wish the combo-box item to create. EditLive! supports various size formats such as **pt**, **%** and **px**.

*Example:* The following combo-box item will display the text at **14pt**.

```
<comboBoxItem name="14pt" text="14pt" />
```

If the above combo-box item is selected, the following tag will be created.

```
<span style="font-size: 14pt"></span>
```

## Optional Attributes

### text

This attribute gives the value which appears inside the relevant combo box within EditLive! (eg. Heading 1, Normal, 12pt, Times New Roman).

In EditLive! 6.0 and above, use of the text attribute is no longer recommended for the style combo box as default names are included along with translations for each supported interface language. All common block tags are included in the list of default names:

- P
- DIV (in 6.4 and above)
- H1-6
- PRE
- ADDRESS
- TD
- TH
- TR
- TABLE
- LI
- UL
- OL
- DD
- DT
- DIR
- MENU
- DL

### Example

The following example adds the H1 style to the Style combo box so that it appears as *Heading 1* inside the combo box in EditLive!. Also added is the Arial font to the Typeface combo box; it is listed as *Company Font* in the combo box. Finally, the HTML font size 3 is added to the Size combo box and listed as *12pt*.

All the combo boxes in this example are added to the Format Toolbar.

```
<editLive>
...
<toolbars>
  <toolbar name="format">
    <toolbarComboBox name="Style">
      <comboBoxItem name="H1" />
    </toolbarComboBox>
    <toolbarComboBox name="Face">
      <comboBoxItem name="Arial" text="Company Font" />
    </toolbarComboBox>
  </toolbar>
</toolbars>
```



```
        </toolbarComboBox>
        <toolbarComboBox name="Size">
            <comboBoxItem name="3" text="12pt"/>
        </toolbarComboBox>
    </toolbar>
</toolbars>
...
</editLive>
```

## Remarks

The `<comboBoxItem>` element can appear multiple times within the `<toolbarComboBox>` element.

The `<comboBoxItem>` element must be a complete tag, it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<comboBoxItem name=... />
```

## **customComboBoxItem**

# customComboBoxItem (Applet)

This element specifies the properties for a developer-defined custom combo box item for use within EditLive!. The custom combo box item must be listed within a `<customToolBarComboBox>` element and will therefore appear on one of the toolbars within EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolBarComboBox>
<customComboBoxItem (Applet)>
```

```
<editLive>
...
<toolbars>
  <toolbar>
    <customToolBarComboBox>
      <customComboBoxItem ... />
    </customToolBarComboBox>
  </toolbar>
</toolbars>
...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom combo box item within the `<customToolBarComboBox>` element. This means that there cannot be two `<customComboBoxItem>` elements with the same name within one `<customToolBarComboBox>` element.

### text

The text to represent this item within the combo box it is to be listed in.

### action

The action which this custom combo box item performs when selected. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.  
The specified HTML must already be URL encoded.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.  
The specified HTML must already be URL encoded.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string.
- *PostDocument* - Post the content of the applet to a server-side script.

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the **action** attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call. The function will be called providing the id of the editor that the item was selected on. This id is the value used in the JavaScript constructor or in the case of inline mode the div id of the editable section.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! for Java uses to POST its content.  
This parameter is required.
  - *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java is to POST to.  
The parameter is required.

- *Response Processing*  
The operation that EditLive! for Java is to perform with the HTTP response from the POST acceptor script.  
The parameter can have the following values:
  - *saveToDisk* - Present the user with a save file dialog, with which they can save the response to the local machine.
  - *callback* - Pass the entire content of the HTTP response to a specified JavaScript callback function for processing.

This parameter is required.

- *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.

This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the value attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the `insertHTMLAtCursor` action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3C/p%3E`.

## Conditional Attributes

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

The **enableintag** attribute is required when using the *customPropertiesDialog* action. The *enableintag* attribute will not work with any of the other action attributes.

## Examples

The following example demonstrates how to define a custom combo box item for use within a custom combo box which exists on the EditLive! Command Toolbar. The combo box item defined in this example will insert HTML to insert at the cursor. Note that the value in the example below is [URL encoded](#).

```
<editLive>
...
<toolbars>
  <toolbar name="command">
    <customToolbarComboBox name="customCombo">
      <customComboBoxItem
        name="customComboItem1"
        text="Custom Combo Item"
        action="insertHTMLAtCursor"
        value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
      </customToolbarComboBox>
    </toolbar>
  </toolbars>
...
</editLive>
```

The following example demonstrates how to define a custom combo box item which uses the *raiseEvent* action for use within a custom combo box which exists on the EditLive! Command Toolbar. The combo box item defined in this example will call the JavaScript function called *eventRaised*. This JavaScript function will show an alert with the text "button clicked on " with the id of the editor when then combo box item is selected.

```
function eventRaised(id) {
    alert("button clicked on " + id);
}
```

```
<editLive>
...
toolbars>
  <toolbar name="command">
    <customToolbarComboBox name="customCombo">
      <customComboBoxItem
```

```
        name="customComboItem1"  
        text="Custom Combo Item"  
        action="raiseEvent "  
        value="eventRaised" />  
    </customToolBarComboBox>  
</toolbar>  
</toolbars>  
    ...  
</editLive>
```

## Remarks

The `<customComboBoxItem>` element can appear multiple times within the `<customToolBarComboBox>` element.

The `<customComboBoxItem>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customComboBoxItem name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

## See Also

- [Encoding Content for Use with EditLive!](#)
- [Creating Custom Menu and Toolbar Items](#)

# customComboBoxItem (Swing SDK)

This element specifies the properties for a developer-defined custom combo box item for use within EditLive! for Java Swing. The custom combo box item must be listed within a `<customToolBarComboBox>` element and will therefore appear on one of the toolbars within EditLive! for Java Swing.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolBarComboBox>
<customComboBoxItem>
```

```
<editLive>
...
<toolbars>
  <toolbar>
    <customToolBarComboBox>
      <customComboBoxItem ... />
    </customToolBarComboBox>
  </toolbar>
</toolbars>
...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom combo box item within the `<customToolBarComboBox>` element. This means that there cannot be two `<customComboBoxItem>` elements with the same name within one `<customToolBarComboBox>` element.

### text

The text to represent this item within the combo box it is to be listed in.

### action

The action which this custom combo box item performs when selected. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
  - The specified HTML must already be URL encoded.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
  - The specified HTML must already be URL encoded.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the [ELJBean](#) will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the [ELJBean](#) will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *PostDocument* - Post the content of the applet to a server side script.

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the *action* attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! for Java Swing uses to POST its content.  
  
This parameter is required.
  - *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java Swing for Java is to POST to.  
  
The parameter is required.
  - *Response Processing*  
The operation that EditLive! for Java Swing is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
    1. *saveToDisk* - Presents the user with a save file dialog, with which they can save the response to the local machine.
    2. *callback* - Passes the entire content of the HTTP response to a specified JavaScript callback function for processing.  
  
This parameter is required.
  - *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.  
  
This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the *value* attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! for Java Swing in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called `_JSFunction_`.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be URL encoded in the XML file. For example, `<p>HTML to insert<p>` becomes `<p>%3Cp%3EHTML%20to%20insert%3C/p%3E`.

### Example

The following example demonstrates how to define a custom combo box item for use within a custom combo box which exists on the EditLive! for Java Swing Command Toolbar. The combo box item defined in this example will insert HTML to insert at the cursor. Note that the value in the example below is URL encoded.

```
<editLive>
  ...
  <toolbars>
    <toolbar name="command">
      <customToolbarComboBox name="customCombo">
        <customComboBoxItem
          name="customComboItem1"
          text="Custom Combo Item"
          action="insertHTMLAtCursor"
          value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
        </customToolbarComboBox>
      </toolbar>
    </toolbars>
  ...
</editLive>
```

### Remarks

The `<customComboBoxItem>` element can appear multiple times within the `<customToolbarComboBox>` element.

The `<customComboBoxItem>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customComboBoxItem name=... />
```

# customMenuItem



# customMenuItem (Applet)

This element specifies the properties for a developer-defined custom menu item for use within EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<customMenuItem (Applet)>
```

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <customMenuItem... />
    </menu>
  </menuBar>
  ...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom menu item.

### text

The text to place on the menu for this item.

### action

The action which this custom action performs. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. The function will be called providing the id of the editor that the item was clicked on. This id is the value used in the JavaScript constructor or in the case of inline mode the div id of the editable section.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string.
- *PostDocument* - Post the content of the applet to a server side script.

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the action attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string ##ephox##. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! for Java uses to POST its content.  
This parameter is required.
  - *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java is to POST to.  
This parameter is required.
  - *Response Processing*  
The operation that EditLive! for Java is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
    - *saveToDisk* - Present the user with a save file dialog, with which they can save the response to the local machine.
    - *callback* - Pass the entire content of th HTTP response to a specified JavaScript callback function for processing.  
This parameter is required.

- *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.

This `callback` parameter should only be used if the response processing is set to *callback*.

The parameters specified through the value attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the `insertHTMLAtCursor` action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3Cp%3E`.

## Conditional Attributes

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to `td` the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

The `enableintag` attribute is required when using the `customPropertiesDialog` action. The `enableintag` attribute will not work with any of the other action attributes.

## Optional Attributes

### imageURL

The URL of the image to be placed on the menu with the menu item text. The image should be of a `.gif` format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide. This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! is embedded.

You can also explicitly set mnemonics and shortcuts for this element using the [Mnemonic and Shortcut Attributes](#).

### designViewOnly

This attribute is used to indicate the action is only applicable to the design view of EditLive!. When set to true, the custom button (or menu item) will be disabled when the editor is switched to code view.

The default value is false.

## Examples

The following example demonstrates how to define a custom menu item for use within EditLive!. The menu item defined in this example will insert HTML at the cursor. Note that the value in the example below is URL encoded.

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="Example">
      <customMenuItem
        name="customItem1"
        text="Custom Item"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="insertHTMLAtCursor"
        value="%3Cp%3EHTML%20to%20insert%3Cp%3E" />
      </menu>
    ...
  </menuBar>
  ...
</editLive>
```

The following example demonstrates how to define a custom menu item which uses the `raiseEvent` action for use within EditLive!. The menu item defined in this example will call the JavaScript function called `eventRaised`. This JavaScript function will show an alert with the text "button clicked on " with the id of the editor when then menu item is clicked.

```
function eventRaised(id) {
    alert("button clicked on " + id);
}
```

```
<editLive>
...
<menuBar>
...
<menu name="ephox_insertmenu">
  <customMenuItem
    name="customItem1"
    text="Raise Event"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="raiseEvent"
    value="eventRaised" />
  </menu>
...
</menuBar>
...
</editLive>
```

The following example demonstrates how to define a custom menu item which uses the *customPropertiesDialog* action for use within EditLive!. The menu item defined in this example will call the JavaScript function called *DisplayAttributes*. This menu item is only available when a `<h1>` tag is selected.

This operation will only work for one specified tag type. To specify which tag type the menu or toolbar item will appear for, use the **enableintag** attribute.

```
<editLive>
...
<menuBar>
...
<menu name="ephox_insertmenu">
  <customMenuItem
    name="showAttributes"
    text="H1 Properties"
    action="customPropertiesDialog"
    value="DisplayAttributes"
    enableintag="h1"
  />
</menu>
...
</menuBar>
...
</editLive>
```

For an instance of EditLive! using the `<customMenuItem>` created above, the following function would create a JavaScript dialog displaying each name-value pair of attributes for the `<h1>` tag.

```
function DisplayAttributes(properties)
{
    alert("VALUE-NAME pairs: " + properties);
}
```

The following example demonstrates how to define a custom menu item which uses the *PostDocument* action for use within EditLive!. The menu item defined in this example will POST the content in the field *editlive\_field* to the script at <http://someserver/post/POSTacceptor.aspx>. Upon completion of the POST, the content of the HTTP response will be passed to the JavaScript callback function *JSFunction*.

```
<editLive>
...
<menuBar>
...
<menu>
  <customMenuItem
    name="customItem1"
    text="POST Content"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="PostDocument" value="editlive_field##ephox##http://someserver/post/POSTacceptor.aspx"
```

```
##ephox##callback##ephox##JSFunction" />
  </menu>
  ...
</menuBar>
...
</editLive>
```

## Remarks

The **<customMenuItem>** element can appear multiple times within the **<menu>** element.

The **<customMenuItem>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customMenuItem name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

## See Also

- [Encoding Content for Use with EditLive!](#)
- [Creating Custom Menu and Toolbar Items](#)

# customMenuItem (Swing SDK)

This element specifies the properties for a developer-defined custom menu item for use within EditLive! for Java Swing.

## Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<customMenuItem>
```

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <customMenuItem... />
    </menu>
  </menuBar>
  ...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom menu item.

### text

The text to place on the menu for this item.

### action

The action which this custom action performs. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the `ELJBean` will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the `ELJBean` will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *PostDocument* - Post the content of the applet to a server side script.

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the *action* attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:

- *Post Field*  
The name of the field in the HTTP POST that EditLive! uses to POST its content.  
This parameter is required.
- *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java is to POST to.  
The parameter is required.
- *Response Processing*  
The operation that EditLive! is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
  1. *saveToDisk* - Presents the user with a save file dialog, with which they can save the response to the local machine.
  2. *callback* - Passes the entire content of the HTTP response to a specified JavaScript callback function for processing.  
This parameter is required.
- *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.  
This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the *value* attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the *value* attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to *http://someserver/postacceptor.jsp*, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, <p>HTML to insert<p> becomes %3Cp%3EHTML%20to%20insert%3Cp%3E.

## Conditional Attributes

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a <td> tag (i.e. a table cell).

The **enableintag** attribute is required when using the *customPropertiesDialog* action. The **enableintag** attribute will not work with any of the other **action** attributes.

## Optional Attributes

### imageURL

The URL of the image to be placed on the menu with the menu item text. The image should be of a .gif format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide. This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! for Java Swing is embedded.

You can also explicitly set mnemonics and shortcuts for this element using the [Mnemonic and Shortcut Attributes](#).

### designViewOnly

This attribute is used to indicate the action is only applicable to the design view of EditLive! for Java Swing. When set to true, the custom button (or menu item) will be disabled when the editor is switched to code view.

The default value is false.

## Examples

The following example demonstrates how to define a custom menu item for use within EditLive! for Java Swing. The menu item defined in this example will insert HTML at the cursor. Note that the value in the example below is URL encoded.

```
<editLive>
...
<menuBar>
...
<menu name="Example">
  <customMenuItem
    name="customItem1"
    text="Custom Item"
```

```

        imageURL="http://www.someserver.com/imagel6x16.gif"
        action="insertHTMLAtCursor"
        value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </menu>
    ...
</menuBar>
...
</editLive>

```

The following example demonstrates how to define a custom menu item which uses the *raiseEvent* action for use within EditLive! for Java Swing. The menu item defined in this example will call the JavaScript function called *eventRaised*.

```

<editLive>
...
<menuBar>
...
<menu name="ephox_insertmenu">
    <customMenuItem
        name="customItem1"
        text="Raise Event"
        imageURL="http://www.someserver.com/imagel6x16.gif"
        action="raiseEvent"
        value="eventRaised" />
    </menu>
...
</menuBar>
...
</editLive>

```

The following example demonstrates how to define a custom menu item which uses the *customPropertiesDialog* action for use within EditLive! for Java Swing. The menu item defined in this example will call the JavaScript function called *DisplayAttributes*. This menu item is only available when a `<h1>` tag is selected.

This operation will only work for one specified tag type. To specify which tag type the menu or toolbar item will appear for, use the **enableintag** attribute.

```

<editLive>
...
<menuBar>
...
<menu name="ephox_insertmenu">
    <customMenuItem
        name="showAttributes"
        text="H1 Properties"
        action="customPropertiesDialog"
        value="DisplayAttributes"
        enableintag="h1"
    />
</menu>
...
</menuBar>
...
</editLive>

```

For an instance of EditLive! using the `<customMenuItem>` created above, the following function would create a JavaScript dialog displaying each name-value pair of attributes for the `<h1>` tag.

```

function DisplayAttributes(properties)
{
    alert("VALUE-NAME pairs: " + properties);
}

```

The following example demonstrates how to define a custom menu item which uses the *PostDocument* action for use within EditLive!. The menu item defined in this example will POST the content in the field *editlive\_field* to the script at *http://someserver/post/POSTacceptor.aspx*. Upon completion of the POST, the content of the HTTP response will be passed to the JavaScript callback function *JSFunction*.

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <customMenuItem
        name="customItem1"
        text="POST Content"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="PostDocument" value="editlive_field##ephox##http://someserver/post/POSTacceptor.aspx
##ephox##callback##ephox##JSFunction" />
      </menu>
    ...
  </menuBar>
  ...
</editLive>
```

## Remarks

The `<customMenuItem>` element can appear multiple times within the `<menu>` element.

The `<customMenuItem>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customMenuItem name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

## See Also

- [Creating Custom Menu and Toolbar Items](#)



# customTags

This element allows you to set options relating to custom tags, both registered and unknown, within EditLive!.

## Configuration Element Tree Structure

`<editLive>`  
`<wysiwygEditor>`  
`<customTags>`

```
<editLive>
  ...
  <wysiwygEditor>
    <customTags>
      <!--customTags settings-->
    </customTags>
  </wysiwygEditor/>
  ...
</editLive>
```

## Child Elements

`<doubleClickActions>`

This element allows a list of actions to be configured for use with a mouse double click.

## Remarks

The `<customTags>` element can appear only once within the `<wysiwygEditor>` element.

## **customToolBarButton**

# customToolBarButton (Applet)

This element will cause a particular button to be present on the toolbar within Ephox EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolBarButton (Applet)>
```

```
<editLive>
  ...
  <toolbars>
    ...
    <toolbar>
      <customToolBarButton... />
    </toolbar>
  </toolbars>
  ...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom toolbar button.

### text

The tooltip text for this custom toolbar button.

### action

The action which this toolbar button performs when clicked on. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. The function will be called providing the id of the editor that the item was clicked on. This id is the value used in the JavaScript constructor or in the case of inline mode the div id of the editable section.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string.
- *PostDocument* - Post the content of the applet to a server side script

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the **action** attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:
  - *Post Field*  
The name of the field in the HTTP POST that EditLive! for Java uses to POST its content.  
This parameter is required.
  - *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java is to POST to.  
This parameter is required.
  - *Response Processing*  
The operation that EditLive! for Java is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
    - *saveToDisk* - Present the user with a save file dialog, with which they can save the response to the local machine.
    - *callback* - Pass the entire content of th HTTP response to a specified JavaScript callback function for processing.  
This parameter is required.

- *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.

This `callback` parameter should only be used if the response processing is set to *callback*.

The parameters specified through the **value** attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to <http://someserver/postacceptor.jsp>, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be **URL encoded** in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3Cp%3E`.

## Conditional Attributes

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

The **enableintag** attribute is required when using the *customPropertiesDialog* action. The **enableintag** attribute will not work with any of the other **action** attributes.

## Optional Attributes

### imageUrl

The URL of the image to be placed on the toolbar button. The image should be of a .gif format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide. This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! is embedded.

### designViewOnly

This attribute is used to indicate the action is only applicable to the design view of EditLive!. When set to true, the custom button (or menu item) will be disabled when the editor is switched to code view.

The default value is false.

## Examples

The following example demonstrates how to define a custom toolbar button for use within EditLive!. The toolbar button defined in this example will insert HTML to insert at the cursor. Note that the value in the example below is URL encoded.

```
<editLive>
  ...
  <toolbars>
    ...
    <toolbar name="Example">
      <customToolbarButton
        name="customItem1"
        text="Custom Item"
        imageUrl="http://www.someserver.com/image16x16.gif"
        action="insertHTMLAtCursor"
        value="%3Cp%3EHTML%20to%20insert%3Cp%3E" />
      </toolbar>
    ...
  </toolbars>
  ...
</editLive>
```

The following example demonstrates how to define a custom toolbar button which uses the *raiseEvent* action for use within EditLive!. The toolbar button defined in this example will call the JavaScript function called *eventRaised*. This JavaScript function will show an alert with the text "button clicked on " with the id of the editor when then menu item is clicked.

```
function eventRaised(id) {
  alert("button clicked on " + id);
}
```

```

<editLive>
  ...
  <toolbars>
    ...
    <toolbar name="ephox_insertmenu">
      <customToolbarButton
        name="customItem1"
        text="Raise Event"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="raiseEvent"
        value="eventRaised" />
    </toolbar>
    ...
  </toolbars>
  ...
</editLive>

```

The following example demonstrates how to define a custom menu item which uses the *customPropertiesDialog* action for use within EditLive!. The toolbar button defined in this example will call the JavaScript function called *DisplayAttributes*. This toolbar button is only available when a `<h1>` tag is selected.

This operation will only work for one specified tag type. To specify which tag type the menu or toolbar item will appear for, use the **enableintag** attribute.

```

<editLive>
  ...
  <toolbars>
    ...
    <toolbar name="ephox_insertmenu">
      <customToolbarButton
        name="showAttributes"
        text="H1 Properties"
        action="customPropertiesDialog"
        value="DisplayAttributes"
        enableintag="h1"
      />
    </toolbar>
    ...
  </toolbars>
  ...
</editLive>

```

For an instance of EditLive! using the **<customToolbarButton>** created above, the following function would create a JavaScript dialog displaying each name-value pair of attributes for the `<h1>` tag.

```

function DisplayAttributes(properties)
{
  alert("VALUE-NAME pairs: " + properties);
}

```

The following example demonstrates how to define a custom toolbar button which uses the **PostDocument** action for use within EditLive!. The toolbar button defined in this example will POST the content in the field *editlive\_field* to the script at <http://someserver/post/POSTacceptor.aspx>. Upon completion of the POST, the content of the HTTP response will be passed to the JavaScript callback function *JSFunction*.

```

<editLive>
  ...
  <toolbars>
    ...
    <toolbar name="Example">
      <customToolbarButton
        name="customItem1"
        text="POST Content"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="PostDocument" value="editlive_field##ephox##http://someserver/post/POSTacceptor.aspx
##ephox##callback##ephox##JSFunction" />
    </toolbar>
    ...

```

```
</toolbars>
...
</editLive>
```

## Remarks

The `<customToolBarButton>` element can appear multiple times within the `<toolbar>` element.

The `<customToolBarButton>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customToolBarButton name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

## See Also

- [Encoding Content for Use with EditLive!](#)
- [Creating Custom Menu and Toolbar Items](#)

# customToolBarButton (Swing SDK)

This element will cause a particular button to be present on the toolbar within Ephox EditLive! for Java Swing for Java Swing.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolBarButton>
```

```
<editLive>
  ...
  <toolbars>
    ...
    <toolbar>
      <customToolBarButton... />
    </toolbar>
  </toolbars>
  ...
</editLive>
```

## Required Attributes

### name

The name which uniquely defines this custom toolbar button.

### text

The tooltip text for this custom toolbar button.

### action

The action which this toolbar button performs when clicked on. This attribute has the following possible values:

- *insertHTMLAtCursor* - Insert the given HTML at the cursor.
- *insertHyperlinkAtCursor* - Insert the given hyperlink at the cursor.
- *raiseEvent* - Call a JavaScript function with the name specified in the value attribute. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the `ELJBean` will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *customPropertiesDialog* - Call a JavaScript function with the name specified in the value attribute. The current tag's properties are also passed to this function as a string. An event is also fired to the [ELJBean](#) using this XML configuration file. The following values will be assigned to the `TextEvent` sent with the event:
  - Action Command - `TextEvent.CUSTOM_ACTION`
  - Extra String - the string specified in the value attribute
  - Extra Object - A Java Map containing the attributes of the element selected when the specified tag is double clicked. The event sent to the `ELJBean` will also have the value `TextEvent.CustomAction.RAISE_EVENT` added to the extra int property of the event. When the event has been handled (by using the event method `setHandled(boolean b)`, where `b = true`), the JavaScript function defined in value will be called.
- *PostDocument* - Post the content of the applet to a server side script.

For more information on these actions, please see the [Creating Custom Menu and Toolbar Items](#) article.

### value

The value of this attribute depends on the value specified in the *action* attribute.

- *insertHTMLAtCursor* - value will be a string of HTML.
- *insertHyperlinkAtCursor* - value will be a URL.
- *raiseEvent* - value will be the name of the JavaScript function to call.
- *customPropertiesDialog* - value will be the name of the JavaScript function to call, passing the current tag's attributes as a string.
- *PostDocument* - the value attribute is used to specify several different parameters. Each parameter is delimited with the string `##ephox##`. The following are the different parameters that can be specified through the value attribute:

- *Post Field*  
The name of the field in the HTTP POST that EditLive! for Java Swing for Java Swing uses to POST its content.  
This parameter is required.
- *Post Acceptor URL*  
The URL for the POST acceptor that EditLive! for Java Swing for Java Swing for Java is to POST to.  
The parameter is required.
- *Response Processing*  
The operation that EditLive! for Java Swing for Java Swing is to perform with the HTTP response from the POST acceptor script. The parameter can have the following values:
  1. *saveToDisk* - Presents the user with a save file dialog, with which they can save the response to the local machine.
  2. *callback* - Passes the entire content of the HTTP response to a specified JavaScript callback function for processing.  
This parameter is required.
- *JavaScript Callback Function*  
The name of the JavaScript callback function to use for processing the response.  
This parameter should only be used if the response processing is set to *callback*.

The parameters specified through the *value* attribute string must appear in the order Post Field, Post Acceptor URL, Response Processing, and JavaScript Callback Function (if needed).

### Example

The following parameters specified through the value attribute string would store the contents of EditLive! for Java Swing for Java Swing in a hidden HTML form field called *POST\_field*, sending the contents via HTTP Post to *http://someserver/postacceptor.jsp*, then call back the JavaScript function called *JSFunction*.

```
value="POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunction"
```

When using the *insertHTMLAtCursor* action the HTML to be inserted must be [URL encoded](#) in the XML file. For example, `<p>HTML to insert<p>` becomes `%3Cp%3EHTML%20to%20insert%3Cp%3E`.

## Conditional Attributes

### enableintag

This attribute defines in which tags the function should be enabled. For example, when set to *td* the function will be enabled when the cursor is within a `<td>` tag (i.e. a table cell).

The **enableintag** attribute is required when using the *customPropertiesDialog* action. The **enableintag** attribute will not work with any of the other **action** attributes.

## Optional Attributes

### imageURL

The URL of the image to be placed on the toolbar button. The image should be of a .gif format and be a size of sixteen (16) pixels high and sixteen (16) pixels wide. This URL can be relative or absolute. Relative URLs are relative to the location of the page in which EditLive! for Java Swing is embedded.

### designViewOnly

This attribute is used to indicate the action is only applicable to the design view of EditLive! for Java Swing. When set to true, the custom button (or menu item) will be disabled when the editor is switched to code view.

The default value is false.

## Examples

The following example demonstrates how to define a custom toolbar button for use within EditLive! for Java Swing. The toolbar button defined in this example will insert HTML to insert at the cursor. Note that the value in the example below is URL encoded.

```
<editLive>
...
<toolbars>
...
<toolbar name="Example">
  <customToolbarButton
    name="customItem1"
    text="Custom Item"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="insertHTMLAtCursor"
```



```

        value="%3Cp%3EHTML%20to%20insert%3C/p%3E" />
    </toolbar>
    ...
</toolbars>
...
</editLive>

```

The following example demonstrates how to define a custom toolbar button which uses the *raiseEvent* action for use within EditLive! for Java Swing. The toolbar button defined in this example will call the JavaScript function called *eventRaised*.

```

<editLive>
...
<toolbars>
...
  <toolbar name="ephox_insertmenu">
    <customToolbarButton
      name="customItem1"
      text="Raise Event"
      imageURL="http://www.someserver.com/image16x16.gif"
      action="raiseEvent"
      value="eventRaised" />
    </toolbar>
  ...
</toolbars>
...
</editLive>

```

The following example demonstrates how to define a custom menu item which uses the *customPropertiesDialog* action for use within EditLive! for Java Swing. The toolbar button defined in this example will call the JavaScript function called *DisplayAttributes*. This toolbar button is only available when a `<h1>` tag is selected.

This operation will only work for one specified tag type. To specify which tag type the menu or toolbar item will appear for, use the **enableintag** attribute.

```

<editLive>
...
<toolbars>
...
  <toolbar name="ephox_insertmenu">
    <customToolbarButton
      name="showAttributes"
      text="H1 Properties"
      action="customPropertiesDialog"
      value="DisplayAttributes"
      enableintag="h1"
    />
  </toolbar>
  ...
</toolbars>
...
</editLive>

```

For an instance of EditLive! for Java Swing using the `<customToolbarButton>` created above, the following function would create a JavaScript dialog displaying each name-value pair of attributes for the `<h1>` tag.

```

function DisplayAttributes(properties)
{
  alert("VALUE-NAME pairs: " + properties);
}

```

The following example demonstrates how to define a custom toolbar button which uses the **PostDocument** action for use within EditLive! for Java Swing. The toolbar button defined in this example will POST the content in the field *editlive\_field* to the script at *http://someserver/post/POSTacceptor.aspx*. Upon completion of the POST, the content of the HTTP response will be passed to the JavaScript callback function *JSFunction*.

```

<editLive>
...
<toolbars>

```

```
...
<toolbar name="Example">
  <customToolbarButton
    name="customItem1"
    text="POST Content"
    imageURL="http://www.someserver.com/image16x16.gif"
    action="PostDocument" value="editlive_field##ephox##http://someserver/post/POSTacceptor.aspx
##ephox##callback##ephox##JSFunction" />
  </toolbar>
  ...
</toolbars>
...
</editLive>
```

## Remarks

The `<customToolbarButton>` element can appear multiple times within the `<toolbar>` element.

The `<customToolbarButton>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<customToolbarButton name=... />
```

Text assigned to the value attribute must be URL encoded as it is in the example above.

## See Also

- [Creating Custom Menu and Toolbar Items](#)

# customToolBarComboBox

This element specifies the properties for a developer defined custom toolbar combo box for use within Tiny EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<customToolBarComboBox>
```

```
<editLive>
...
<toolbars>
  <toolbar>
    <customToolBarComboBox>
      <!--custom toolbar combo box settings-->
    </customToolBarComboBox>
  </toolbar>
</toolbars>
...
</editLive>
```

## Required Attributes

name

The name which uniquely defines this custom toolbar combo box.

text

Since *EditLive 9.0.2.57*

The text to display as a title for the combo box. This text is displayed when the list is collapsed.

## Child Elements

```
<customComboBoxItem>
```

This element defines an item which is to be used within a custom combo box.

## Example

The following example demonstrates how to define a custom toolbar combo box for use within EditLive! on the Command Toolbar.

```
<editLive>
...
<toolbars>
  <toolbar name="command">
    <customToolBarComboBox name="CustomCombo">
      <!--customToolBarComboBox settings-->
    </customToolBarComboBox>
  </toolbar name="command">
</toolbars>
...
</editLive>
```

## Remarks

The `<customToolBarComboBox>` element can appear multiple times within the `<toolbar>` element.

# document

This element allows for the configuration of information which is contained between the <HEAD> tags and as attributes within the <BODY> tag of the view within Tiny EditLive!. This includes any <META> information and style sheet links.

## Configuration Element Tree Structure

<editLive>  
<document>

```
<editLive>
  <document>
    <!-- document configuration settings -->
  </document>
  ...
</editLive>
```

## Child Elements

<html>

This structure allows for the configuration of information which is contained between the <HEAD> and as attributes within the <BODY> tag of the EditLive! document. This includes any <META> information and style sheet links.

## Remarks

The structure of the child elements for the document element have been designed so as to resemble the structure of a HTML document. The <html> element provides no more configuration information than that of the document element. It is present in the XML configuration document only to maintain the resemblance of the <document> element structure to that of a HTML document.

The <document> element can appear only once within the <editLive> element.

# doubleClickActions

This element allows you to set options for double click actions related to the interface element specified in the parent tag.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<customTags>  
<doubleClickActions>
```

```
<editLive>  
  ...  
  <wysiwygEditor>  
    <customTags>  
      <doubleClickActions>  
        <!--doubleClickActions settings-->  
      </doubleClickActions>  
    </customTags>  
  </wysiwygEditor/>  
  ...  
</editLive>
```

## Child Elements

```
<action>
```

This element allows an action to be specified when a double click occurs in a specified empty tag.

## Remarks

The `<customTags>` element can appear only once within the `<wysiwygEditor>` element.

# editLive

This is the root element to be used in every EditLive! configuration file.

## Configuration Element Tree Structure

<editLive>

```
<editLive>
  <!-- configuration settings -->
</editLive>
```

## Child Elements

<document>

The document structure handles the configuration of information placed inside the <HEAD> and <BODY> tags. This also includes <META> information and style sheet links.

<ephoxLicenses>

The ephoxLicenses element handles the configuration of licensing information. Licensing information is used to ensure that you are using EditLive! correctly. Tiny will provide you with the appropriate license files; these simply have to be added here.

<spellCheck (Applet)>

The spellCheck element allows for the specification of the spell checker .jar file. This enables customization of the spell checker.

<htmlFilter>

The htmlFilter element allows various filtering options to be set. The HTML will be "cleaned" according to the filtering attributes contained within this element.

<sourceEditor>

The sourceEditor element allows you to set options relating to the EditLive! Code View.

<wysiwygEditor>

This element allows you to set options relating to the EditLive! editing pane.

<wordImport>

The wordImport attribute allows for the configuration of the Microsoft Word import format and the way that Microsoft Word style information is imported.

<mediaSettings>

The mediaSettings structure allows for the configuration of image settings. It provides a mechanism for the developer to provide a list of server images to be made available to end users, as well as configuring how EditLive! behaves with respect to images on both the server and local machines.

<authentication> - **Deprecated**

The authentication structure allows for the configuration of authentication information for use with EditLive!. The username and password settings present within this element will be used when retrieving data from the specified realms.

<hyperlinks>

The hyperlinks structure allows the provision of a list of hyperlinks to the user. These settings affect the hyperlink dialog within EditLive!.

<menuBar>

The menuBar structure allows for the configuration of the EditLive! menus.

This structure configures only the menu bar and not the shortcut menu.

<toolbars>

The toolbars structure allows for the configuration of the EditLive! format and command toolbars.

<shortcutMenu>

The shortcutMenu element allows for the customization of the EditLive! shortcut menu.

## Remarks

Each EditLive! configuration file must contain exactly one <editLive> element.

# ephoxLicenses

This structure contains the various licenses for Tiny EditLive!. If this structure is left blank, or if the licensing information contained within this element is incorrect, then EditLive! will only run in the 30 day trial mode.

## Configuration Element Tree Structure

`<editLive>`  
`<ephoxLicenses>`

```
<editLive>
  ...
  <ephoxLicenses>
    <!--ephoxLicenses configuration settings-->
  </ephoxLicenses>
  ...
</editLive>
```

## Child Elements

`<license>`

This element contains configuration information for an individual license for EditLive!.

## Example

This example demonstrates how to use an empty `<ephoxLicenses>` element to run EditLive! in the 30 day trial mode only.

```
<editLive>
  ...
  <ephoxLicenses/>
  ...
</editLive>
```

## Remarks

The `<ephoxLicenses>` element can appear only once within the `<editLive>` element.

If the `<ephoxLicenses>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<ephoxLicenses/>
```

If left blank then EditLive! will only run in the 30 day trial mode.



# excelImport

This element configures the manner in which EditLive! reacts when text is imported from Microsoft Excel.

## Configuration Element Tree Structure

`<editLive>`  
`<excelImport>`

```
<editLive>
  ...
  <excelImport ... />
  ...
</editLive>
```

## Optional Attributes

### styleOption

This attribute specifies the user prompting and behaviour of EditLive! upon detecting an import from Microsoft Excel. This attribute has four possible values:

- *user\_prompt* - Users will be prompted as to whether they wish to import Microsoft Excel styles as inline or embedded styles, or strip them from the imported text. The dialog used to prompt the user is the same as the Paste Special dialog.
- *merge\_inline\_styles* - This setting will result in EditLive! importing styles from Microsoft Excel with style information applied inline in place of class attributes.
- *clean* - The clean setting will result in EditLive! stripping the Microsoft Excel styles from the imported text; only structural HTML elements will be imported, such as `<b>`, `<p>`, etc.
- *plain\_text* - The *plain\_text* setting will strip out all style information from the imported text.

Styles imported from Microsoft Excel will not overwrite styles which already exist within the document.

### cleanOption

This boolean value specifies whether the *Clean HTML* option will appear in the **Paste Special...** dialog.

Default value: *true*

### mergeInlineStylesOption

This boolean value specifies whether the *Styled HTML (Inline)* option will appear in the **Paste Special...** dialog.

Default value: *true*

### plainTextOption

This boolean value specifies whether the *Plain Text* option will appear in the **Paste Special...** dialog.

Default value: *true*

## Example

The following example demonstrates how to set EditLive! to prompt the user with style import options every time a Microsoft Excel import is detected.

```
<editLive>
  ...
  <excelImport styleOption="user_prompt" />
  ...
</editLive>
```

## Remarks

The `<excelImport>` element can appear only once within the `<editLive>` element.

If the `<excelImport>` element is to be left blank then the element must be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<excelImport styleOption=... />
```



# head

This element allows for the configuration of information which is to be contained between the <HEAD> tags of the EditLive! document. Settings which are configured through the <head> element will appear in the actual document inside EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<document>  
<html>  
<head>
```

```
<editLive>  
  <document>  
    <html>  
      <head>  
        <!--head configuration settings-->  
      </head>  
      . . . .  
    </html>  
  </document>  
  . . .  
</editLive>
```

## Child Elements

```
<base>
```

This element provides the information which is to be stored in the <BASE> tag within the <HEAD> tag of the EditLive! document.

```
<meta>
```

This element provides the information which is to be stored in the <META> tag within the <HEAD> tag of the EditLive! document.

```
<link>
```

This element provides the style sheet information which is to be stored in a <LINK> tag within the <HEAD> tag of the EditLive! document.

This element defaults to a style sheet information type (ie. text/css information).

```
<style>
```

This element provides the information which is to be stored between the <STYLE> tags, between the <HEAD> tags of the Tiny EditLive! document. This element has no attributes.

```
<title>
```

This element provides the style sheet information which is to be stored in a <LINK> tag within the <HEAD> tag of the EditLive! document.

This element defaults to a style sheet information type (i.e. text/css information).

## Remarks

The <head> element can appear only once within the <html> element.

# html

This element allows for the configuration of information which is contained between the <HEAD> tags and as attributes within the <BODY> tag of the EditLive! document. This includes any META information and style sheet links.

## Configuration Element Tree Structure

```
<editLive>  
<document>  
<html>
```

```
<editLive>  
  <document>  
    <html>  
      <!--html configuration settings-->  
    </html>  
  </document>  
  ...  
</editLive>
```

## Child Elements

```
<head>
```

This structure allows for the configuration of information which is to be contained between the <HEAD> tags of the EditLive! document.

```
<body>
```

This structure allows for the configuration of the attributes of the <BODY> tag of the EditLive! document.

## Remarks

The extra level in the XML document tree provided by this element serves no practical purpose beyond that of the document element. No attributes are set within the <html> element itself. This element has been added to the tree as a conceptual aide only. Through this element the concept of the HTML document can be better maintained and visualized.

The <html> element can appear only once within the <document> element.

# htmlFilter

This element provides the HTML filter settings for use within EditLive!

## Configuration Element Tree Structure

`<editLive>`  
`<htmlFilter>`

```
<editLive>
  ...
  <htmlFilter ... />
  ...
</editLive>
```

## Required Attributes

### wrapLength

Specifies the number of characters for each line within the HTML source.

Setting the **wrapLength** to zero (0) turns wrapping off within the HTML source.

## Optional Attributes

These attributes are **all** booleans and can only be *true* or *false*.

### indentContent

This option specifies that EditLive! will indent the content of appropriate tags, thus creating well-formatted HTML source code. If left blank the default value will be used.

Default Value: *true*

### logicalEmphasis

If set to *true*, `<B>` tags will be replaced by `<STRONG>` tags and `<I>` tags will be replaced by `<EM>` tags. If left blank the default value will be used.

Default Value: *true*

### numericEntities

If set to *true*, entities such as `&nbsp;` are converted to numeric entities (eg `&#xA0;`). The default value depends on the use of `outputXHTML` or `outputXML`.

Default Value

- When `outputXHTML` or `outputXML` are set to *true*: *true*
- Otherwise: *false*

### outputXHTML

If set to *true*, output is created as XHTML. If left blank the default value will be used. When set to *true*, `numericEntities` is also set to *true* by default.

Default Value: *false*

### outputXML

If set to *true*, output is created as XML. If left blank the default value will be used. When set to *true*, `numericEntities` is also set to *true* by default.

Default Value: *false*

### xhtmlStrict

If set to *true*, the editor removes settings for values which are not allowed in XHTML strict:

- In the "List Properties" dialog, the settings for "start" and "value" are removed
- In the "Hyperlink" dialog, the setting for "target" is removed

Default Value: *false*

### uppercaseTags

If set to *true*, tags will be in uppercase within the source. If left blank the default value will be used.

Default Value: *true*

### quoteMarks

If set to *true*, quotation marks (") will be escaped and therefore appear as ". If left blank the default value will be used.

Default Value: *true*

### encloseText

If set to *true*, content will be automatically be wrapped with paragraph (<p>) tags if it has not been properly enclosed with block tags. This is done to ensure that the content inside EditLive! is valid.

Default Value: *true*

Ephox does not recommend setting this attribute to false as it may cause invalid HTML to be generated.

### allowUnknownTags

If set to *true*, tags not recognised in HTML or XHTML will be interpreted as custom tags inside EditLive!. This will preserve the unknown tags. When set to *false* unknown tags will be HTML encoded (e.g. <unknownTag> would be converted to <UnknownTag>).

Default Value: *true*

When producing XHTML output, allowing unknown tags to be preserved in the content may cause errors in validating the resulting XHTML. If the XHTML produced by EditLive! is to be run through a validation process then it is recommended that this attribute be set to *false*.

### wrapCustomTagsInP

If set to *true*, any custom tag specified outside of a block element will be automatically wrapped in a <P> tag.

Default Value: *true*

## Example

The following example demonstrates how to set the various attributes of the <htmlFilter> element.

```
<editLive>
  ...
  <htmlFilter wrapLength="68" indentContent="true" logicalEmphasis="true" />
  ...
</editLive>
```

## Remarks

To ensure EditLive! provides valid XHTML content where required, ensure that *allowUnknownTags* is set to *false* and *outputXHTML* is set to *true*.

The <htmlFilter> element can appear only once within the <editLive> element.

If the <htmlFilter> element is to be left blank then the element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<htmlFilter wrapLength=... />
```

# htmlImport

This element configures the manner in which Tiny EditLive! reacts when a user attempts to paste HTML into the editor. This includes HTML copied from external webpages or from EditLive! itself.

This does *not* apply for HTML documents imported from Microsoft Word. For configuration settings for importing text from Microsoft Word please see the [wordImport](#) configuration element.

## Configuration Element Tree Structure

```
<editLive>
<htmlImport>
```

```
<editLive>
  ...
  <htmlImport ... />
  ...
</editLive>
```

## Optional Attributes

### styleOption

This attribute specifies the user prompting and behaviour of EditLive! upon detecting imported content. This attribute has five possible values:

- *user\_prompt* - Users will be prompted as to whether they wish to import the styles from their imported text as inline or embedded styles or strip them from the imported text. The dialog used to prompt the user is the same as the Paste Special dialog.
- *merge\_embedded\_styles* - This setting will result in EditLive! importing styles from the selected content with the styles stored in the `<head>` of the document and class attributes will be applied where relevant.
- *merge\_inline\_styles* - This setting will result in EditLive! importing styles from the selected content with style information applied inline in place of class attributes.
- *clean* - The clean setting will result in EditLive! stripping the styles from the imported text. Only structural HTML elements will be imported, such as `<b>`, `<p>`, etc.
- *plain\_text* - The *plain\_text* setting will strip out all style information from the imported text.

Styles imported from content will not overwrite styles which already exist within the document.

### cleanOption

This boolean value specifies whether the *Clean HTML* option will appear in the **Paste Special...** dialog.

Default value: *true*

### mergeInlineStylesOption

This boolean value specifies whether the *Styled HTML (Inline)* option will appear in the **Paste Special...** dialog.

Default value: *true*

### mergeEmbeddedStylesOption

This boolean value specifies whether the *Styled HTML (Embedded)* option will appear in the **Paste Special...** dialog.

Default value: *true*

### plainTextOption

This boolean value specifies whether the *Plain Text* option will appear in the **Paste Special...** dialog.

Default value: *true*

## Example

The following example demonstrates how to set EditLive! to prompt the user with style import options every time a HTML content import is detected.

```
<editLive>
  ...
  <htmlImport styleOption="user_prompt" />
  ...
</editLive>
```

---

## Remarks

The `<htmlImport>` element can appear only once within the `<editLive>` element.

If the `<htmlImport>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<htmlImport styleOption=.../>
```



# httpUpload

This element allows for the configuration of information used when using HTTP upload within Tiny EditLive! to upload embedded image and media files.

## Configuration Element Tree Structure

```
<editLive>
<mediaSettings>
<httpUpload>
```

```
<editLive>
  ...
  <mediaSettings>
    <httpUpload ... />
  </mediaSettings>
  ...
</editLive>
```

## Required Attributes

These attributes are ignored when a repository child element is used.

### base

This attribute defines the location where images can be found after they have been uploaded. Each local image URL will be replaced with this URL so that images will be accessed from the server location. For more information please see the [developer guide article on HTTP upload](#).

### href

This attribute defines the location on the Web server of the script which handles image uploads. For more information please see the [developer guide article on HTTP upload](#).

## Optional Attributes

### allowObjectUpload

This attribute defines whether local object files are automatically uploaded using the **base** and **href** attributes mentioned above.

This attribute is a boolean, and can have the value of either *true* or *false*.

### uploadFileFieldName

This attribute defines the field name for the upload file. This attribute is a string.

## Child Elements

### <httpUploadData>

This element is used to provide extra information when performing a HTTP upload. This can be used so that extra information is provided to your HTTP upload handler script via HTTP Header elements. Multiple elements are allowed; data is specified in name and value pairings.

<httpUploadData> elements are included in regular HTTP uploads.

### <httpPostData>

This element is used to provide extra information when performing a HTTP upload. This can be used so that extra information is provided to your HTTP upload handler script via HTTP POST Elements. Multiple elements are allowed; data is specified in name and value pairings.

<httpPostData> elements are included in regular HTTP uploads.

### <repository> - **Deprecated**

Overrides the **base** and **href** attributes of the <httpUpload> tag and instead uploads all media to a WebDAV server.

## Example

The following example demonstrates how to define the base and href attributes for EditLive!.

```
<editLive>
  ...
```

```
<mediaSettings>
  <httpUpload
    base="http://yourserver.com/imagedir/"
    href="http://yourserver.com/scripts/uploadhandler.asp"
  </httpUpload>
</mediaSettings>
...
</editLive>
```

## Remarks

The **<httpUpload>** element can appear only once within the **<mediaSettings>** element.

If there are no **<httpUploadData>** or **<repository>** elements then **<httpUpload>** must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<httpUpload base=... />
```

If there are **<httpUploadData>**, **<httpPostData>** or **<repository>** elements present then the **<httpUpload>** element needs to have both opening and closing tags. It should appear as below:

```
...
<httpUpload base=... >
  <httpUploadData name=... />
  <httpUploadData name=... />
</httpUpload>
...
```

## See Also

- [HTTP Upload Support for Images and Objects](#)

# httpUploadData

This element allows for the configuration of information used when using HTTP media upload within EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<httpUpload>  
<httpUploadData>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <httpUpload ... >  
      <httpUploadData ... />  
    </httpUpload>  
  </mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

name

This attribute should contain the name for the extra HTTP Header property being transmitted with the HTTP upload.

data

This attribute should contain the extra data that you wish to transmit with the HTTP upload.

## Example

The following example demonstrates how to set various `<httpUploadData>` attributes.

```
<editLive>  
  ...  
  <mediaSettings>  
    <httpUpload  
      base="http://yourserver.com/mediaDir/"  
      href="http://yourserver.com/scripts/uploadhandler.asp"  
    >  
      <httpUploadData name="user" data="default"/>  
      <httpUploadData name="priority" data="1"/>  
      ...  
    </httpUpload>  
  </mediaSettings>  
  ...  
</editLive>
```

## Remarks

The `<httpUploadData>` element can appear multiple times within the `<httpUpload>` element.

The `<httpUploadData>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<httpUploadData name=... />
```

# httpPostData

This element allows for the configuration of information used when using HTTP media upload within EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<httpUpload>  
<httpPostData>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <httpUpload ... >  
      <httpPostData ... />  
    </httpUpload>  
  </mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

name

This attribute should contain the name for extra HTTP POST form data to transmitted with the HTTP upload.

data

This attribute should contain the form data that you wish to transmit with the HTTP upload.

## Example

The following example demonstrates how to set various **<httpPostData>** attributes.

```
<editLive>  
  ...  
  <mediaSettings>  
    <httpUpload  
      base="http://yourserver.com/mediaDir/"  
      href="http://yourserver.com/scripts/uploadhandler.asp"  
    >  
      <httpPostData name="user" data="default" />  
      <httpPostData name="priority" data="1" />  
      ...  
    </httpUpload>  
  </mediaSettings>  
  ...  
</editLive>
```

## Remarks

The **<httpPostData>** element can appear multiple times within the **<httpUpload>** element.

The **<httpPostData>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<httpPostData name=... />
```

# hyperlink

This element allows for the specification of a single hyperlink that the end users of EditLive! will be provided with via the Insert Hyperlink dialog.

## Configuration Element Tree Structure

```
<editLive>  
<hyperlinks>  
<hyperlinkList>  
<hyperlink>
```

```
<editLive>  
  ...  
  <hyperlinks>  
    <hyperlinkList>  
      <hyperlink ... />  
    </hyperlinkList>  
  </hyperlinks>  
  ...  
</editLive>
```

## Required Attributes

href

This attribute defines the URL for the hyperlink.

## Optional Attributes

description

This attribute specifies the description used for the image in the *Insert Hyperlink* dialog within EditLive!.

target

This attribute has the same effect as the target property of the <A> HTML tag. This attribute specifies the name of the frame for the hyperlink to jump to. When inserting the hyperlink defined by this **<hyperlink>** element into an EditLive! document, this attribute will appear in the HTML source code.

title

This attribute has the same effect as the title property of the <A> HTML tag. This attribute provides an advisory title for the document linked to. When inserting the hyperlink defined by this **<hyperlink>** element into an EditLive! document, this attribute will appear in the HTML source code.

## Example

The following example demonstrates how to specify a hyperlink to provide the users of EditLive! with.

```
<editLive>  
  ...  
  <hyperlinks>  
    <hyperlinkList>  
      <hyperlink href="http://www.someserver.com/somepage.html"  
        description="This is a hyperlink."  
        target="_blank"  
        title="Hyperlink" />  
    </hyperlinkList>  
  </hyperlinks>  
  ...  
</editLive>
```

## Remarks

The **<hyperlink>** element can appear multiple times within the **<hyperlinkList>** element.

The **<hyperlink>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<hyperlink href=... />
```

# hyperlinkList

This element allows for the specification of a list of hyperlinks that the users of EditLive! will be provided with via the Insert Hyperlink dialog.

## Configuration Element Tree Structure

```
<editLive>  
<hyperlinks>  
<hyperlinkList>
```

```
<editLive>  
  ...  
  <hyperlinks>  
    <hyperlinkList>  
      <!--hyperlink list configuration settings-->  
    </hyperlinkList>  
  </hyperlinks>  
  ...  
</editLive>
```

## Optional Attributes

`enabled`

This boolean attribute specifies whether the hyperlink dialog displays the option for users to link to existing files or web pages. This attribute is a boolean with possible values of *true* or *false*.

Default Value: *true*

## Child Elements

```
<hyperlink>
```

This element defines a hyperlink that users of EditLive! will be provided with via the Insert Hyperlink dialog.

## Remarks

The `<hyperlinkList>` element can appear only once within the `<hyperlinks>` element.

# hyperlinks

This element allows for the configuration of a list of hyperlinks and e-mail addresses to be made available to the end users of EditLive! via the Insert Hyperlink dialog.

## Configuration Element Tree Structure

[<editLive>](#)  
[<hyperlinks>](#)

```
<editLive>
  <document>
    <!--document configuration settings-->
  </document>
  ...
</editLive>
```

## Child Elements

[<hyperlinkList>](#)

This element allows for the specification of the list of hyperlinks that the end users will be provided with. It also allows the "Existing File of Web Page" panel of the hyperlink dialog to be hidden.

[<mailtoList>](#)

This element allows for the specification of the list of e-mail addresses that the end users will be provided with. It also allows the "email address" panel of the hyperlink dialog to be hidden.

[<placesInDocumentList>](#)

This element allows the "Places in Document" panel of the hyperlink dialog to be hidden.

[<webdav>](#) - **Deprecated**

This element allows for the specification of WebDAV repositories that end users can browse and select files to link to.

## Remarks

The [<hyperlinks>](#) element can appear only once within the [<editLive>](#) element.



# image

This element defines the properties of an image stored on a server which is to be used with Ephox EditLive!.

## Configuration File Element

```
<editLive>  
<mediaSettings>  
<images>  
<imageList>  
<image>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <images>  
      ...  
        <imageList>  
          <image ... />  
        </imageList>  
      </images>  
    </mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

src

This attribute defines where the image can be found.

The URL can be absolute or relative. If a relative URL is specified it will be relative to the base attribute defined in the [<httpUpload>](#) element. If no base attribute has been defined and the URL is relative then the URL will be relative to the base of the page in which the instance of EditLive! resides.

## Optional Attributes

align

This attribute has the same effect as the *align* property of the <IMG> HTML tag. It affects the alignment of text which is placed after the image reference. When inserting the image defined by this **<image>** element into an EditLive! document this attribute will appear in the source code.

alt

This attribute has the same effect as the *alt* property of the <IMG> HTML tag. This attribute defines the alternative text to be displayed when the image cannot be loaded into a HTML page. When inserting the image defined by this **<image>** element into an EditLive! document this attribute will appear in the source code.

border

This attribute has the same effect as the *border* property of the <IMG> HTML tag. This attribute specifies the width of the border, in pixels, around the image. When inserting the image defined by this **<image>** element into an EditLive! document this attribute will appear in the source code.

description

This attribute specifies the description used for the image in the Server Image Dialog within EditLive!.

height

This attribute has the same effect as the *height* property of the <IMG> HTML tag. This attribute specifies the height of the image in pixels. When inserting the image defined by this **<image>** element into an EditLive! document this attribute will appear in the source code.

hspace

This attribute has the same effect as the *hspace* property of the <IMG> HTML tag. This attribute specifies the horizontal spacing around the image in pixels (ie. left and right side padding). When inserting the image defined by this **<image>** element into an EditLive! document this attribute will appear in the source code.

name

This attribute has the same effect as the *title* property of the <IMG> HTML tag. This attribute defines the name for the image. When inserting the image defined by this <image> element into an EditLive! document this attribute will appear in the source code.

#### title

This attribute specifies the title used for the image in the Server Image Dialog within EditLive!.

#### vspace

This attribute has the same effect as the *vspace* property of the <IMG> HTML tag. This attribute specifies the vertical spacing around the image in pixels (ie. top and bottom padding). When inserting the image defined by this <image> element into an EditLive! document this attribute will appear in the source code.

#### width

This attribute has the same effect as the *width* property of the <IMG> HTML tag. This attribute specifies the width of the image in pixels. When inserting the image defined by this <image> element into an EditLive! document this attribute will appear in the source code.

## Example

The following example demonstrates how to configure a server image for use with EditLive!.

```
<editLive>
  ...
  <mediaSettings>
    <images>
      ...
      <imageList>
        <image align="left"
          alt="Alternative Text"
          border="1"
          description="A Server Image"
          height="500" width="250"
          src="http://yourserver.com/"
        />
      ...
    </imageList>
  </images>
</mediaSettings>
  ...
</editLive>
```

## Remarks

The <image> element can appear multiple times within the <imageList> element.

The <image> element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<image src=... />
```

# imageBrowser

This element allows for the specification of the image insertion web page component of the image insertion dialog within EditLive!. This allows a webpage to be shown that contains links to various images. Multiple `<imageBrowser>` elements can be included to provide access to multiple pages, or a single browser can be used with links to each page.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<images>  
<imageBrowser>
```

```
<editlive>  
  <mediasettings>  
    <images>  
      <imageBrowser.../>  
    </images>  
  </mediasettings>  
</editlive>
```

## Required Attributes

### href

The location of the web page to display.

This may either be a relative or absolute URL defining the location of the image insertion web page. If this is a relative URL then it defines the location of the image insertion web page relative to the document root directory of the instance of EditLive! concerned.

### forceAbsolute

This attribute specifies whether the location of an image will be represented by its absolute pathname or the path relative to where the instance of EditLive! is being called. This attribute is a boolean and can have the value of either *true* or *false*.

For example, if a web page specified in the `<href>` attribute contains the following line of code:

```
<a href="images/ephoxImage.gif">An Ephox Image</a>
```

Using relative referencing, the URL of the image being inserted into EditLive! would be:

```

```

If using absolute referencing, the URL of the image being inserted would in this format:

```

```

If all instances of EditLive! are operating on the same server as the location of the images, **forceAbsolute** can be set to *false*. Otherwise, it is advised to set **forceAbsolute** to *true*.

## Optional Attributes

### name

The name that appears below the button image.

The `default` button name is translated for each EditLive! interface translation. Use of this attribute will override it for all languages.

### imageURL

The location of an image to display instead of the default Image Browser button. Note that EditLive! will not resize this image, so if it is too big it will expand to the borders of the button and hide the text. For dialog consistency, Tiny recommends using the same size as the existing buttons (32x32).

## Example

The following example demonstrates how to define the image insertion web page for an instance of EditLive!. This example sets the button name to "Tiny Website" and retains the EditLive! default Image Browser button image.

```
<editLive>
  ...
  <images>
    <imageBrowser href="http://tiny.cloud" forceAbsolute="false" name="Tiny Website" />
  </images>
  ...
</editLive>
```

## Remarks

Each **<imageBrowser>** element within the **<images>** element will appear as a new button on the dialog. Buttons will appear in the same order as they are written in the configuration file.

# imageDialog

This element allows developers to customize the appearance of the Image Insertion dialog for EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<images>  
<imageDialog>
```

```
<editlive>  
  <mediasettings>  
    <images>  
      <imageDialog.../>  
    </images>  
  </mediasettings>  
</editlive>
```

## Required Attributes

The following attributes are defined based on factors such as the editor's width and height and the screen resolution for the current user.

### width

The width, in pixels, of the Image Insertion dialog.

### height

The height, in pixels, of the Image Insertion dialog.

### split

The width, in pixels, of the pane next to the preview image pane.

### Example

When selecting the Image Library option in the Image Insertion Dialog, a list of available images will appear. If the split is set to 200 pixels, this list would have a width of 200 pixels.

## Example

The following example demonstrates how to define appearance of the Image Insertion dialog for EditLive!.

```
<editLive>  
  ...  
  <images>  
    <imageDialog width="900" height="400" split="600"/>  
  </images>  
  ...  
</editLive>
```

## Remarks

The `<imageDialog>` element can only appear once within the `<images>` element.

# imageList

## Description

This element contains a list of all the server images to be configured for use with EditLive!.

## Configuration Element Tree Structure

[<editLive>](#)  
[<mediaSettings>](#)  
[<images>](#)  
[<imageList>](#)

```
<editLive>
  ...
  <mediaSettings>
    <images>
      ...
      <imageList>
        <!--image list configuration settings-->
      </imageList>
    </images>
  </mediaSettings>
  ...
</editLive>
```

## Child Elements

[<image>](#)

This element contains the configuration information for a server image for use with EditLive!.

## Remarks

The [<imageList>](#) element can appear only once within the [<images>](#) element.

# images

This element allows for the configuration of information for images and any other media within EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<images>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <images>  
      <!--images configuration settings-->  
    </images>  
  </mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

### allowLocalImages

This attribute defines whether users have the option to browse their local directories for images in the image dialog box. This attribute may be set to *true* or *false*. If set to *false*, users cannot browse local images.

To turn off local image browsing, the *Insert Local Image...* menu item must also be absent from the configuration.

### allowUserSpecified

This attribute defines whether users have the option to specify URLs for images in the image dialog box. This attribute may be set to *true* or *false*. If set to *false*, users cannot specify image their own URLs.

If a user specifies the URL for a local file then EditLive! will attempt to upload this file.

## Optional Attributes

The following *optional* attributes will only work if the **imageEditor** plugin is used in the **<Plugins>** element.

### preferredWidth

This attribute defines the preferred width for local images. If the user inserts a local image with a width larger than **preferredWidth**, EditLive! will resize the image so it has a maximum width of **preferredWidth** while keeping the aspect ratio constant. The enforcement is dependent on the memory available in the client environment.

### preferredHeight

This attribute defines the preferred height for local images. If the user inserts a local image with a height larger than **preferredHeight**, EditLive! will resize the image so it has a maximum height of **preferredHeight** while keeping the aspect ratio constant. The enforcement is dependent on the memory available in the client environment.

### renameResizedImages

By default, images that are resized due to exceeding the **preferredWidth** and **preferredHeight** settings are renamed as well as resized. Setting this attribute to *false* allows that behaviour to be disabled so that images will retain their original filename when they are resized.

## Child Elements

### <imageList>

This element provides a list of images stored on a Web server which can be accessed by the end users of EditLive!.

### <webdav> - Deprecated

This element allows for the customization of the EditLive! WebDAV functionality.

## Remarks

The **<images>** element can appear only once within the **<mediaSettings>** element.

## See Also

- [HTTP Upload Support for Images and Objects](#)



# inlineToolbar

This element contains the configuration information for an inline toolbar for use within EditLive!. Items will appear in a toolbar in EditLive!, from left to right, in the order that they appear in the EditLive! configuration document.

## Configuration Element Tree Structure

```
<editLive>
<inlineToolbars>
<inlineToolbar>
```

```
<editLive>
...
<inlineToolbars>
  <inlineToolbar ...>
    <!--toolbar configuration settings-->
  </inlineToolbar>
</inlineToolbars>
...
</editLive>
```

## Required Attributes

name

An identifying name for this toolbar. Inline toolbars are generated by Tiny for editing specific HTML elements. The following inline toolbars are currently available for EditLive!:

- *img* - An inline toolbar which will appear when selecting an IMG element in EditLive!.

In order to enable the image inline toolbar, you need to include a `<plugin>` element containing a name attribute with the value **imageEditor**. For more information on EditLive!'s Image Editor functionality, please read the [Image Editing](#) article.

### Example

```
<editlive>
...
<plugins>
  <plugin name="imageEditor" />
</plugins>

  <inlineToolbars>
    <inlineToolbar name="img">
      <!--toolbar configuration settings-->
    </inlineToolbar>
  </inlineToolbars>
</editlive>
```

- *table* - An inline toolbar which will appear when selecting a TABLE element or any of its children elements in EditLive!.

In order to enable the table inline toolbar, you need to include a `<plugin>` element containing a name attribute with the value *tableToolbar*.

### Example

```
<editlive>
...
<plugins>
  <plugin name="tableToolbar" />
</plugins>

  <inlineToolbars>
    <inlineToolbar name="table">
      <!--toolbar configuration settings-->
    </inlineToolbar>
  </inlineToolbars>
</editlive>
```

## Child Elements

### [<toolbarButton>](#)

This element will cause a particular button to be present on the toolbar within EditLive!.

### [<toolbarButtonGroup>](#)

This element will cause a particular group of buttons to be present on the toolbar within EditLive!. The operation of these buttons within EditLive! will be mutually exclusive. The buttons added by this element can only be added and removed from the toolbar as a group. For example, the alignment buttons are a button group as Align Left cannot be activated at the same time as Align Right.

### [<toolbarComboBox>](#)

This element will cause a particular combo box to be present on the toolbar within EditLive!.

### [<toolbarSeparator>](#)

This element will cause the appearance of a vertical separating line between toolbar elements.

### [<customToolbarButton>](#)

This element specifies the properties for a developer defined custom toolbar button for use within EditLive!.

### [<customToolbarComboBox>](#)

This element specifies the properties for a developer defined custom toolbar combo box for use within EditLive!.

## Example

This example demonstrates how to declare inline toolbars for IMG and TABLE HTML elements.

```
<editLive>
  ...
  <inlineToolbars>
    ...
    <inlineToolbar name="img">
      ...
    </inlineToolbar>
    <inlineToolbar name="table">
      ...
    </inlineToolbar>
    ...
  </inlineToolbars>
  ...
</editLive>
```

## Remarks

The [<inlineToolbar>](#) element can appear multiple times within the [<inlineToolbars>](#) element.

## See Also

- [Image Editing](#)

# inlineToolbars

This element contains the configuration information for the inline toolbars within Ephox EditLive!.

## Configuration Element Tree Structure

[<editLive>](#)  
[<inlineToolbars>](#)

```
<editLive>
  ...
  <inlineToolbars>
    <!--inlineToolbar configuration settings-->
  </inlineToolbars>
  ...
</editLive>
```

## Optional Attributes

### showOnMainToolbar

This boolean attribute defines if the inlineToolbar is docked to the main toolbar or floats above the relevant content. If *true* the toolbar will appear under the existing toolbars defined in the [<toolbars>](#) element. Docked toolbars will appear only when contextually relevant - i.e. when the cursor is within an image or a table.

Default Value: *false*

## Child Elements

[<inlineToolbar>](#)

This element contains the configuration information for a toolbar within EditLive!.

## Remarks

The [<inlineToolbars>](#) element can appear only once within the [<editLive>](#) element.

# license

This element contains the configuration information for a single license of EditLive!.

Attributes within this element should be entered as per the license file provided by Tiny. If the configuration information provided by the attributes does not correspond to a valid license provided by Tiny then EditLive! will only run in 30 day trial mode.

## Configuration Element Tree Structure

```
<editLive>  
<ephoxLicenses>  
<license>
```

```
<editLive>  
  ...  
  <ephoxLicenses>  
    <license ... />  
  </ephoxLicenses>  
  ...  
</editLive>
```

## Required Attributes

domain

This attribute provides the domain to which this copy of EditLive! is licensed.

expiration

This attribute provides the expiration date of the license.

key

This attribute provides the product key for this license of EditLive!.

licensee

This attribute provides the company or organization to which this copy of EditLive! is licensed.

product

This attribute details the Tiny product which can be used with this license.

release

This attribute details the release number of the Tiny product which can be used with this license.

seats

This attribute details the number of seats that this license is valid for.

## Optional Attributes

accountID

This attribute details your Tiny account ID.

activationURL

This attribute configures the URL that EditLive! should use to check its license. If left blank the default value will be used.

Default Value: <http://www.ephox.com/activate/eljf10.asp>

eqEditor

When set to true this attribute allow the Tiny Equation Editor to be used with the instance of EditLive!. This attribute is a boolean and can only be *true* or *false*.

Default Value: *false*

## forceActive

When set to true, this attribute will force licenses to become active instead of requesting the user to activate the license. If left blank the default value will be used. This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

## type

This attribute specifies the type of license provided. Some license types might be time limited. If left blank the default value will be used. This attribute has the possible values of *production*, *development* or *QA*.

Default Value: *production*

## Example

This example demonstrates how to use the `<license>` element to input licensing information into EditLive!.

```
<editLive>
  ...
  <ephoxLicenses>
    <license accountID="1234"
      activationURL="http://www.ephox.com/elregister/el2/activate.asp"
      domain="demo.com" expiration="NEVER" forceActive="false"
      key="4545-5465-2456-5648" licensee="Someone"
      product="EditLive! for Java" release="2.0" seats="100"
      type="production" />
  </ephoxLicenses>
  ...
</editLive>
```

## Remarks

The `<license>` element can appear multiple times within the `<ephoxLicenses>` element.

The `<license>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<license accountID=... />
```

# link

This element provides the information which is to be stored as attributes within the <LINK> tag between the <HEAD> tags of the EditLive! document. The value which appears within the <link> element will appear within the actual EditLive! document between the <HEAD> tags in a <LINK> tag.

## Configuration Element Tree Structure

```
<editLive>  
<document>  
<html>  
<head>  
<link>
```

```
<editLive>  
  <document>  
    <html>  
      <head>  
        ...  
        <link ... />  
      </head>  
      ...  
    </html>  
  </document>  
  ...  
</editLive>
```

## Optional Attributes

href

This attribute specifies the value for the **href** attribute of the <LINK> tag to be used between the <HEAD> tags within the actual EditLive! document. The **href** attribute specifies the destination for the link (eg. the URL of a stylesheet).

type

This attribute specifies the value for the **type** attribute of the <LINK> tag to be used between the <HEAD> tags within the actual EditLive! document. The **type** attribute specifies the data type for the document or resource which is linked to (eg. text/css).

Default Value: *text/css*

rel

This attribute specifies the value for the **rel** attribute of the <LINK> tag to be used between the <HEAD> tags within the actual EditLive! document. The **rel** attribute specifies relationship between the current document and the link (eg. stylesheet).

## Example

This example demonstrates how to use the <link> element in order to set the attributes of the <LINK> tag within an EditLive! document.

```
<editLive>  
  <document>  
    <html>  
      <head>  
        ...  
        <link href="styles.css" rel="stylesheet" type="text/css" />  
        ...  
      </head>  
      ...  
    </html>  
  </document>  
  ...  
</editLive>
```

## Remarks

The <link> element can appear multiple times within the <head> element. For information on how multiple external style sheets will be interpreted, please refer to the [Using CSS in the Applet](#) article.

The **<link>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<link href=... />
```

## See Also

- [Using CSS in the Applet](#)

# mailto:link

This element allows for the specification of a single hyperlink that the end users of Tiny EditLive! will be provided with via the Insert Hyperlink dialog.

## Configuration Element Tree Structure

```
<editLive>  
<hyperlinks>  
<mailtoList>  
<mailto:link>
```

```
<editLive>  
  ...  
  <hyperlinks>  
    ...  
    <mailtoList>  
      <mailto:link ... />  
    </mailtoList>  
  </hyperlinks>  
  ...  
</editLive>
```

## Required Attributes

href

This attribute defines the e-mail address for the mailto link.

## Optional Attributes

description

This attribute specifies the description used for the mailto link in the Insert Hyperlink dialog within EditLive!.

## Example

The following example demonstrates how to specify an e-mail address to provide the end users of EditLive! with.

```
<editLive>  
  ...  
  <hyperlinks>  
    ...  
    <mailtoList>  
      <mailto:link href="someone@mailserver.com"  
        description="This is a mailto link" />  
    </mailtoList>  
  </hyperlinks>  
  ...  
</editLive>
```

## Remarks

The **<mailto:link>** element can appear multiple times within the **<mailtoList>** element.

The **<mailto:link>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<mailto:link href=... />
```



# mailtoList

This element allows for the specification of a list of e-mail addresses that the end users of EditLive! will be provided with via the Insert Hyperlink dialog.

## Configuration Element Tree Structure

[<editLive>](#)  
[<hyperlinks>](#)  
[<mailtoList>](#)

```
<editLive>
  ...
  <hyperlinks>
    ...
    <mailtoList>
      <!--mailto list configuration settings-->
    </mailtoList>
  </hyperlinks>
  ...
</editLive>
```

## Optional Attributes

enabled

This boolean attribute specifies whether the hyperlink dialog displays the option for users to link to email addresses. This attribute is a boolean with possible values of *true* or *false*.

Default Value: *true*

## Child Elements

[<mailtoLink>](#)

This element defines an e-mail address that end users of EditLive! will be provided with via the Insert Hyperlink dialog.

## Remarks

The [<mailtoList>](#) element can appear only once within the [<hyperlinks>](#) element.

# mathml

This element allows for the configuration of MathML content generated by the [Equation Editor](#) component of the editor.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<mathml>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <mathml ... />  
  </mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

### createEquationImage

This attribute specifies whether equations created in the MathFlow equation editor are stored as either `<img>` tags or MathML markup language. This attribute is a boolean and can only be *true* or *false*. If set to *true*, the `<img>` tag generated will contain a `mathml` attribute. This attribute will contain the `mathml` markup.

Default Value: *false*

## Optional Attributes

### pointSize

This attribute specifies the point size value of the images generated by the [Equation Editor](#). This attribute is expressed as a numerical value.

### dpi

This attribute specifies the dots-per-inch value to be used when printing MathML content. This attribute is expressed as a numerical value.

### defaultFormat

When specifying `createEquationImage=true`, this value specifies what image format is used to represent the MathML. This attribute has two values:

- *gif*
- *png*

Default Value: *png*

*gif* is only available for clients running the editor under JRE 1.6.

## Remarks

The `<mathml>` element can appear only once within the `<mediaSettings>` element.

## See Also

- [<webeqLicense>](#) Configuration File Element

# mediaSettings

This element allows for the configuration of media settings, such as image and object settings, within EditLive!.

## Configuration Element Tree Structure

`<editLive>`  
`<mediaSettings>`

```
<editLive>
  <mediaSettings>
    <!--mediaSettings configuration settings-->
  </mediaSettings>
  ...
</editLive>
```

## Optional Attributes

`defaultFormat`

This attribute specifies the file format images are saved as when an actual image is generated by EditLive!. Some examples of situations where EditLive! actually generates images are:

- When an image is copied from an image editing tool (e.g. Microsoft Paint) and pasted directly into EditLive!.
- When using the Ephox [Equation Editor](#) and storing equations as images. For more information on storing equations as images, see the `<mathml>` configuration element.

This attribute has two possible values:

- *png*
- *jpg*

## Child Elements

`<httpUpload>`

This element allows for the configuration of upload settings for images and embedded objects within the EditLive! content.

`<images>`

This element allows for the configuration of information for images within EditLive!.

`<imageBrowser>`

This element allows for the specification of a webpage to display when using the image insertion dialog.

`<multimedia>`

This element allows for the configuration of settings for using embedded multimedia objects, inserted via the `<OBJECT>` tag, (e.g. Flash and other multimedia content) within EditLive!.

`<mathml>`

This element allows users to specify how equations created using the WebEQ equation editor are stored in EditLive!.

## Remarks

The `<mediaSettings>` element can appear only once within the `<editLive>` element.

# menu

This element contains settings for a specific menu (eg. View, Edit). These settings appear as a list of commands which appear on the menu.

## Configuration Element Tree Structure

```
<editLive>  
<menuBar>  
<menu>
```

```
<editLive>  
  ...  
  <menuBar>  
    <menu>  
      <!--menu configuration settings-->  
    </menu>  
  </menuBar>  
  ...  
</editLive>
```

## Required Attributes

name

This attribute specifies the name of the menu (e.g. Edit, View). EditLive! provides 9 default menu names, or developers can create their own. These default menu names are automatically internationalized to match the user's locale. To create an instance of one of the default menus, use one of the following strings as a value for name:

- *ephox\_filemenu* - The internationalized File menu.
- *ephox\_editmenu* - The internationalized Edit menu.
- *ephox\_viewmenu* - The internationalized View menu.
- *ephox\_insertmenu* - The internationalized Insert menu.
- *ephox\_formatmenu* - The internationalized Format menu.
- *ephox\_toolsmenu* - The internationalized Tools menu.
- *ephox\_tablemenu* - The internationalized Table menu.
- *ephox\_formmenu* - The internationalized Form menu.
- *ephox\_trackchangesmenu* - The internationalized Track Changes menu.
- *ephox\_help* - The internationalized Help menu.

## Optional Attributes

You can explicitly set mnemonics and shortcuts for this element using the [Mnemonic and Shortcut Attributes](#).

## Child Elements

```
<menuItem>
```

This element contains information for an item on the menu (eg. Cut, Undo, Table Properties).

```
<menuItemGroup>
```

This element contains information for a grouping on the menu. The commands added by this element can only be added and removed from the menu as a group.

A grouping is a set of two or more items which are related and their selection is mutually exclusive within EditLive!. For example, the Source View and Design View commands exist in a `<menuItemGroup>`.

```
<menuSeparator>
```

This element informs EditLive! that it should include a horizontal line, or menu separator, within the menu.

```
<customMenuItem>
```

This element specifies the properties for a developer-defined custom menu item for use within EditLive!.

```
<submenu>
```

This element contains information for a submenu item which may be placed within a menu. The Font, Font Size, and Style `<submenu>`s are examples of this.

## Example

The following example demonstrates how to create an instance of the default Edit menu.

```
<editLive>
  ...
  <menuBar>
    <menu name="ephox_editmenu">
      ...
    </menu>
  </menuBar>
  ...
</editLive>
```

## Remarks

The `<menu>` element can appear multiple times within the `<menuBar>` element.

# menuBar

The configuration information contained within this element contains the various settings to use for the menu bar within EditLive!. Items will appear in the menu bar of EditLive!, from left to right, in the order that they appear in the EditLive! configuration document.

The configuration information within this element includes settings for each menu (eg. View, Edit).

## Configuration Element Tree Structure

`<editLive>`  
`<menuBar>`


```
<editLive>
  ...
  <menuBar showAboutMenu='true'>
    <!--menu bar configuration settings-->
  </menuBar>
  ...
</editLive>
```

## Optional Attributes

### showAboutMenu

Boolean. Defaults to false. Determines whether an "About EditLive" menu appears.

Important

 If you remove the About EditLive! menu by setting this attribute to **false** or removing the menuBar element completely you must either use the *helpgroup* megamenu on the toolbar to make the About EditLive! dialog available or replicate the open source licensing information contained within the About EditLive! dialog in another place within your software and/or documentation.

## Child Elements

`<menu>`

This element contains the settings for a specific menu (eg. View, Edit).

## Remarks

The `<menuBar>` element can appear only once within the `<editLive>` element.

# menuItem

This element specifies an item to include within a menu in EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<menuItem>
```

```
<editLive>
...
<menuBar>
  <menu>
    <menuItem ... />
  </menu>
</menuBar>
...
</editLive>
```

## Required Attributes

name

This attribute gives the name for the menu item. For use within a `<menu>` element it must be from the [Menu and Toolbar Item List](#).

In `<submenu>` items this attribute provides the value of the attribute. It should correspond to the name or size of the relevant font or the value for the style. The value used for this attribute will be inserted into the HTML source of the document when the submenu item is selected.

## Optional Attributes

text

This attribute customizes the menu command text.

imageUrl

This attribute changes the image associated with a menu item.

You can also explicitly set mnemonics and shortcuts for this element using the [Mnemonic and Shortcut Attributes](#).

## Examples

The following example demonstrates how to add the `mnuUndo`, `mnuRedo`, `mnuCut`, `mnuPaste`, `mnuSelectAll`, and `mnuFind` items to the Edit menu. Thus the instance of EditLive! using this configuration will have only an Edit menu with these items.

```
<editLive>
...
<menuBar>
  <menu name="Edit">
    <menuItem name="Undo" />
    <menuItem name="Redo" />
    <menuItem name="Cut" />
    <menuItem name="Paste" />
    <menuItem name="SelectAll" />
    <menuItem name="Find" />
  </menu>
</menuBar>
...
</editLive>
```

The following example demonstrates how to add the *Times New Roman*, *Courier New*, and *Arial* fonts to the `mnuFontFace` `<submenu>`. They will be listed as the *New Roman*, *Courier*, and *Company Default* fonts respectively in the submenu due to their text attributes.

```
<editLive>
...
```

```
<menuBar>
  <menu name="Format">
    <submenu name="FontFace">
      <menuItem name="Times New Roman" text="New Roman"/>
      <menuItem name="Courier New" text="Courier"/>
      <menuItem name="Arial" text="Company Default"/>
    </submenu>
  </menu>
</menuBar>
...
</editLive>
```

## Remarks

The `<menuItem>` element can appear multiple times within the `<menu>` element.

The `<menuItem>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<menuItem name=... />
```

## See Also

- `<submenu>` Configuration File Element



# menuItemGroup

This element contains information for a grouping on the menu. A grouping is a set of two or more items which are related and their selection is mutually exclusive within EditLive!. For example, the Source View and Design View commands exist in a <menuItemGroup>. The commands added by this element can only be added and removed from the menu as a group.

## Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<menuItemGroup>
```

```
<editLive>
  ...
  <menuBar>
    <menu>
      <menuItemGroup ... />
    </menu>
  </menuBar>
  ...
</editLive>
```

## Required Attributes

name

This attribute gives the name for the command group. It must be a command from the [Menu and Toolbar Item List](#).

## Examples

The following example demonstrates how to add the Design View and Code View commands to the menu bar.

```
<editLive>
  ...
  <menuBar>
    <menu name="View">
      <menuItemGroup name="SourceView" />
    </menu>
  </menuBar>
  ...
</editLive>
```

## Remarks

The <menuItemGroup> element can appear multiple times within the <menu> element.

The <menuItemGroup> element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<menuItemGroup name=... />
```

# menuSeparator

## Description

This element places a horizontal separating line between commands within a EditLive! menu bar. This line will appear between the commands defined by the `<menuItem>` elements immediately before and after the `<menuSeparator>` element.

This element has no attributes or child elements.

## Configuration Element Tree Structure

```
<editLive>  
<menuBar>  
<menu>  
<menuSeparator />
```

```
<editLive>  
  ...  
  <menuBar>  
    <menu>  
      <menuSeparator />  
    </menu>  
  </menuBar>  
  ...  
</editLive>
```

## Example

The following example demonstrates how to insert a menu separator into a menu. In the instance of EditLive! created from this configuration, the menu separator will appear between the Redo and the Cut commands.

```
<editLive>  
  ...  
  <menuBar>  
    <menu name="Edit">  
      <menuItem name="Undo" />  
      <menuItem name="Redo" />  
      <menuSeparator />  
      <menuItem name="Cut" />  
      <menuItem name="Copy" />  
      <menuItem name="Paste" />  
    </menu>  
  </menuBar>  
  ...  
</editLive>
```

## Remarks

The `<menuSeparator>` element can appear multiple times within the `<menu>` element.

The `<menuSeparator>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<menuSeparator />
```

# meta

This element provides the information which is to be stored as attributes within the <META> tag(s) between the <HEAD> tags of the EditLive! document. The value which appears within the <meta> element will appear within the actual EditLive! document between the <HEAD> tags in a <META> tag.

## Configuration Element Tree Structure

```
<editLive>
<document>
<html>
<head>
<meta>
```

```
<editLive>
  <document>
    <html>
      <head>
        ...
        <meta ... />
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Required Attributes

content

This attribute specifies the value for the content attribute of the <META> tag to be used between the <HEAD> tags within the actual EditLive! document.

## Optional Attributes

http-equiv

This attribute specifies the value for the http-equiv attribute of the <META> tag to be used between the <HEAD> tags within the actual EditLive! document.

name

This attribute specifies the value for the name attribute of the <META> tag to be used between the <HEAD> tags within the actual EditLive! document.

## Example

The following example demonstrates how to specify two different <META> tags for use within EditLive! documents.

```
<editLive>
  <document>
    <html>
      <head>
        ...
        <meta content="text/html; charset=UTF-8"
              http-equiv="Content-Type" name="contentType" />
        <meta name="Author" content="John Doe" />
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Remarks

The **<meta>** element is most often used to specify the character set to be used within EditLive!. This is done by using a **<meta>** element which specifies a charset such as the following example which specifies the UTF-8 character set:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

The **<meta>** element can appear only once within the **<head>** element.

The **<meta>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<meta content=... />
```

# multimedia

This element allows for the configuration of information for multimedia file types embedded via the <OBJECT> tag within Tiny EditLive!

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<multimedia>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <multimedia>  
      <!--multimedia configuration settings-->  
    </multimedia>  
  </mediaSettings>  
  ...  
</editLive>
```

## Optional Attributes

### allowMediaServices

Introduced in EditLive 9.0.1.39

This option specifies whether the *Media Services* tab is visible in the *Insert Media* dialog.

This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

### allowEmbedCode

Introduced in EditLive 9.0.0.130

This option specifies whether the *Embed Code* tab is visible in the *Insert Media* dialog.

This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

### allowHtml5Audio

This option specifies whether the *Audio* tab is visible in the *Insert Media* dialog. The *Audio* tab can be disabled for systems that are not HTML5 compatible, or where this functionality is not required.

This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

### allowHtml5Video

This option specifies whether the *Video* tab is visible in the *Insert Media* dialog. The *Video* tab can be disabled for systems that are not HTML5 compatible, or where this functionality is not required.

This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

## Child Elements

### <services>

This element allows a list of supported oEmbed services to be specified.

### <types>

This element allows a list of supported multimedia file types to be specified.

### <webdav> - Deprecated

This element allows for the customization of the EditLive! WebDAV functionality.

## Remarks

The `<multimedia>` element can appear only once within the `<mediaSettings>` element.

# otherLicenses

## Description

This element allows you to specify licenses for third party products incorporated into EditLive!.

## Configuration Element Tree Structure

[<editLive>](#)  
[<otherLicenses>](#)

```
<editLive>
  ...
  <otherLicenses>
    ...
  </otherLicenses>
  ...
</editLive>
```

## Child Elements

[<webeqLicense>](#)

This element allows OEM vendors incorporating the editor to specify their licensing information for using Design Science MathML products.

# param

This element defines a <param> tag with a name attribute which can be used to configure the multimedia object within EditLive!. Users will be able to set the value of the parameter, and the name and (user specified) value pair will be inserted as a <param> tag for the relevant <object> in the source in EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<mediaSettings>
<multimedia>
<types>
<type>
<param>
```

```
<editLive>
...
<mediaSettings>
...
<multimedia>
...
<types>
...
<type>
  <param name="..." />
</type>
...
</types>
</multimedia>
</mediaSettings>
...
</editLive>
```

## Required Attributes

name

A unique name for the parameter. The value for the name attribute should be the same as the name attribute for the <param> tag to be inserted with the <object> tag into the source for EditLive!.

## Example

The following example demonstrates how to configure parameters of a multimedia type for use with EditLive!. In this case the media type is a Windows Media Player streaming type with the .asx extension.

```
<editLive>
...
<mediaSettings>
...
<multimedia>
<types>
...
<type
  name="Windows Media (Streaming)"
  type="application/x-mplayer2"
  extension="asx"
  allowCustomParams="true"
  urlParam="fileName"
>
  <param name="animationAtStart" />
  <param name="autoStart" />
  <param name="clickToPlay" />
</type>
...
</types>
</multimedia>
```



```
</mediaSettings>  
...  
</editLive>
```

## Remarks

The **<param>** element can appear multiple times within the **<type>** element.

The **<param>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<param name=... />
```

# placesInDocumentList

This element allows the "Places in Document" panel of the Insert Hyperlink dialog to be hidden.

## Configuration Element Tree Structure

```
<editLive>  
<hyperlinks>  
<placesInDocumentList>
```

```
<editLive>  
  ...  
  <hyperlinks>  
    ...  
    <placesInDocumentList />  
    ...  
  </hyperlinks>  
  ...  
</editLive>
```

## Optional Attributes

`enabled`

This boolean attribute specifies whether the hyperlink dialog displays the option for users to link to places in the current document. This attribute is a boolean with possible values of *true* or *false*.

Default Value: *true*

## Remarks

The `<placesInDocumentList>` element can appear only once within the `<hyperlinks>` element.

# plugins (config)

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

## Description

This element allows you to specify Plugins to be loaded into EditLive!.

## Configuration Element Tree Structure

`<editLive>`  
`<plugins>`

```
<editLive>
  ...
  <plugins>
    ...
  </plugins>
  ...
</editLive>
```

## Child Elements

`<plugin>`

`<plugin>` elements can be used in one of two ways:

1. Directly insert `<plugin>` elements under the `<plugins>` element, containing all `<plugin>` child elements (e.g. `<advancedapis (Applet)>`).
2. Insert an empty `<plugin>` element, specifying its URL attribute to load an external plugin XML file.

# realm

WebDAV support has been removed in EditLive! 9.1

This element contains the settings required for a user to authenticate themselves to a realm on a Web server. EditLive! supports the following forms of Web server authentication:

- Basic
- Digest
- NTLM

## Configuration Element Tree Structure

<editLive>  
<authentication>  
<realm>

```
<editLive>
  ...
  <authentication>
    <realm ... />
  </authentication>
  ...
</editLive>
```

## Required Attributes

realm

The realm for which this authentication information applies. For basic and digest authentication the realm is specified in the Web server configuration; in NTLM authentication the realm is equivalent to the host name.

For NTLM authentication, the **realm** attribute should contain the value of the host to be accessed. For example, if the URL for the protected area was *http://www.yourserver.com/webDAV* and this required NTLM authentication, then the realm attribute would be *www.yourserver.com* (i.e. *realm="www.yourserver.com"*).

## Optional Attributes

domain

The domain on which the specified realm can be accessed.

This attribute is not needed when using either the basic or digest authentication types.

password

The password to be used when accessing the realm.

username

The username to be used when accessing the realm.

## Examples

This example demonstrates how to configure the <realm> element for use with a basic or digest authentication method. In this example the realm being accessed is the protected realm. The username to be used is *EditLive* and the corresponding password is *Ephox*.

```
<editLive>
  ...
  <authentication>
    <realm realm="protected"
      username="EditLive"
      password="Ephox" />
  </authentication>
  ...
</editLive>
```

This example demonstrates how to configure the **<realm>** element for use with a NTLM authentication method. In this example the protected area being accessed is designated by the URL *http://yourserver.com/protected* and thus resides on the *yourserver.com* host which can be found on the intranet network domain. The username to be used is *EditLive* and the corresponding password is *Ephox*.

```
<editLive>
  ...
  <authentication>
    <realm realm="yourserver.com"
      domain="intranet"
      username="EditLive"
      password="Ephox"
    />
  </authentication>
  ...
</editLive>
```

## Remarks

The **<realm>** element can appear multiple times within the **<authentication>** element.

The **<realm>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<realm realm=... />
```

In the case of all the authentication details not being provided in the configuration file, the end user will be prompted for the details required. The details provided in the configuration file, if any, will be supplied to the end user when prompted. For example, if only the username and domain are supplied by the configuration file, the end user will be prompted and will only have to supply the correct password.

# repository

WebDAV support has been removed in EditLive! 9.1

## Description

This element defines the configuration settings for the use of EditLive! with a WebDAV repository. Each possible use of the element is separated to allow maximum flexibility with repository setup.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<multimedia>  
<webdav>  
<repository ... />  
</repository ... />
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <multimedia>  
      ...  
      <webdav>  
        <repository ... />  
      </webdav>  
    </multimedia>  
  </mediaSettings>  
  ...  
</editLive>
```

```
<editLive>  
<mediaSettings>  
<images>  
<webdav>  
<repository ... />  
</repository ... />
```

```
<editLive>  
  ...  
  <mediaSettings>  
    <images>  
      ...  
      <webdav>  
        <repository ... />  
      </webdav>  
    </images>  
  </mediaSettings>  
  ...  
</editLive>
```

```
<editLive>  
<hyperlinks>  
<webdav>  
<repository ... />  
</repository ... />
```

```
<editLive>  
  ...  
  <hyperlinks>  
    ...  
    <webdav>  
      <repository ... />  
    </webdav>  
  </hyperlinks>  
  ...  
</editLive>
```

```
<editLive>
<mediaSettings>
<httpUpload>
<repository ... />
```

```
<editLive>
...
<mediaSettings>
...
  <httpUpload>
    <repository ... />
  </httpUpload>
...
</mediaSettings>
...
</editLive>
```

## Required Attributes

### baseDir

The root directory for the WebDAV repository. Users will not be permitted to browse to any directories of a higher level than that of the value of the **baseDir** attribute.

### webDAVBaseURL

The location of the WebDAV repository relative to the document root of the instance of EditLive! concerned.

This may either be a relative or absolute URL defining the location of the WebDAV repository. If this is a relative URL, then it defines the location of the WebDAV repository relative to the document root directory of the instance of EditLive! concerned.

## Optional Attributes

### username

The username required to gain access to the server containing the WebDAV repository. If not specified and anonymous access is disabled, the user will be prompted for a login.

### password

The password required, in conjunction with the username, to gain access to the server containing the WebDAV repository.

### name

The human-readable name for this WebDAV repository.

### defaultDir

The initial directory that EditLive! is to access on the WebDAV server. If not specified, **baseDir** is used.

### useMimeType

Whether or not to filter files according to their mime type. This attribute has two possible values: *true* or *false*. If this is *false*, the files are filtered according to their file extension.

Default: The default value for this attribute is *true*.

## Example

The following example demonstrates how to define a WebDAV repository for image browsing. It specifies the root URL *http://www.yourserver.com/webDAV* for use with an instance of EditLive! which has the root directory *http://www.yourserver.com/editlive*. It uses a relative URL to define the location of the WebDAV repository.

```
<editLive>
...
<mediaSettings>
  <images>
    <webdav>
      <repository name="Sample"
        baseDir="http://www.yourserver.com/webDAV"
```

```

        defaultDir="SampleDir"
        webDAVBaseUrl=" ../webDAV"
    />
</webdav>
</images>
</mediaSettings>
...
</editLive>

```

The following example demonstrates how to define a WebDAV repository for hyperlink browsing. It specifies the root URL *http://www.yourserver.com/webDAV* for use with an instance of EditLive! running on a server different to that of the WebDAV repository. Thus, an absolute URL must be used.

```

<editLive>
...
<mediaSettings>
  <hyperlinks>
    <webdav>
      <repository name="Sample"
        baseDir="http://www.yourserver.com/webDAV"
        defaultDir="SampleDir"
        webDAVBaseUrl="http://www.yourserver.com/webDAV"
      />
    </webdav>
  </hyperlinks>
</mediaSettings>
...
</editLive>

```

The following example demonstrates how to enable EditLive! to upload local images to a WebDAV repository. Using the code below, a local image with a URL of *file:///C:/Documents%20and%20Settings/DefaultUser/My%20Documents/My%20Pictures/ephoxlogo.gif* would be uploaded to *http://www.yourserver.com/webDAV/ephoxlogo.gif*.

```

<editLive>
...
<mediaSettings>
...
  <httpUpload>
    <repository name="Sample"
      username="admin"
      password="admin"
      baseDir="http://www.yourserver.com/webDAV"
      webDAVBaseUrl="http://www.yourserver.com/webDAV"
    />
  </httpUpload>
</mediaSettings>
...
</editLive>

```

## Remarks

Using WebDAV with [<httpUpload>](#) does not allow the user to choose the upload location on the server. EditLive! will simply upload all images to the default directory.

The **<repository>** element can appear multiple times within each **<webdav>** element. Note that [<httpUpload>](#) does not contain a webdav element and only allows one **<repository>** element - if multiple **<repository>** elements are specified in this case, only the first is used.

The first repository listed within each **<webdav>** element is the default WebDAV repository for that feature (images, hyperlinks, or multimedia).

The **<repository>** element must be an empty XML tag; it cannot contain a body - only attributes. This means the tag must be specified as:

```
<repository name=... />
```



# shortcutMenu

This element contains the configuration information for the shortcut menu within Tiny EditLive!.

## Configuration Element Tree Structure

[<editLive>](#)  
[<shortcutMenu>](#)

```
<editLive>
  ...
  <shortcutMenu>
    <!--shortcut menu configuration settings-->
  </shortcutMenu>
  ...
</editLive>
```

## Child Elements

[<shrtMenu>](#)

This element allows for configuration of the shortcut menu within Tiny EditLive!.

## Remarks

The [<shortcutMenu>](#) element can appear only once within the [<editLive>](#) element.

# shrtMenu

This element allows for the configuration of the shortcut menu within Ephox EditLive!. It is the child element of the [<shortcutMenu>](#) element.

## Configuration Element Tree Structure

```
<editLive>  
<shortcutMenu>  
<shrtMenu>
```

```
<editLive>  
  ...  
  <shortcutMenu>  
    <shrtMenu>  
      <!--short menu configuration settings-->  
    </shrtMenu>  
  </shortcutMenu>  
</editLive>
```

## Child Elements

```
<shrtMenuItem>
```

This element defines command items for the shortcut menu.

```
<shrtMenuSeparator>
```

This element will cause the appearance of a horizontal separating line between shortcut menu commands.

```
<customMenuItem>
```

This element specifies the properties for a developer-defined custom menu item for use within EditLive!.

## Remarks

The [<shrtMenu>](#) element can appear only once within the [<shortcutMenu>](#) element.

# shrtMenuItem

This element defines command items for the shortcut menu within Ephox EditLive!

## Configuration Element Tree Structure

```
<editLive>
<shortcutMenu>
<shrtMenu>
<shrtMenuItem>
```

```
<editLive>
  ...
  <shortcutMenu>
    <shrtMenu>
      <shrtMenuItem ... />
    </shrtMenu>
  </shortcutMenu>
  ...
</editLive>
```

## Required Attributes

name

This attribute gives the name for the menu item. This name must be from the [Menu and Toolbar Item List](#).

## Example

The following example configures the Shortcut Menu of EditLive! so that it contains the **Cut, Copy, Paste, Select All, Image Properties..., Table Properties..., Cell Properties, and Hyperlink Properties...** commands.

```
<editLive>
  ...
  <shortcutMenu>
    <shrtMenu>
      <shrtMenuItem name="Cut" />
      <shrtMenuItem name="Copy" />
      <shrtMenuItem name="Paste" />
      <shrtMenuItem name="SelectAll" />
      <shrtMenuItem name="PropImage" />
      <shrtMenuItem name="PropTable" />
      <shrtMenuItem name="PropCell" />
      <shrtMenuItem name="Hyperlink" />
    </shrtMenu>
  </shortcutMenu>
</editLive>
```

## Remarks

The `<shrtMenuItem>` element can appear multiple times within the `<shrtMenu>` element.

The `<shrtMenuItem>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<shrtMenuItem name=... />
```

# shrtMenuSeparator

This element places a horizontal separating line between commands within the Tiny EditLive! shortcut menu. This line will appear between the commands defined by the `<shrtMenuItem>` elements immediately before and after the `<shrtMenuSeparator>` element.

This element has no attributes or child elements.

## Configuration Element Tree Structure

```
<editLive>  
<shortcutMenu>  
<shrtMenu>  
<shrtMenuSeparator>
```

```
<editLive>  
  ...  
  <shortcutMenu>  
    <shrtMenu>  
      <shrtMenuSeparator />  
    </shrtMenu>  
  </shortcutMenu>  
  ...  
</editLive>
```

## Examples

The following example places a shortcut menu separator between the Paste and the Select All commands in the EditLive! shortcut menu.

```
<editLive>  
  ...  
  <shortcutMenu>  
    <shrtMenu>  
      <shrtMenuItem name="Cut" />  
      <shrtMenuItem name="Copy" />  
      <shrtMenuItem name="Paste" />  
      <shrtMenuSeparator />  
      <shrtMenuItem name="SelectAll" />  
    </shrtMenu>  
  </shortcutMenu>  
</editLive>
```

## Remarks

The `<shrtMenuSeparator>` element can appear multiple times within the `<shrtMenu>` element.

The `<shrtMenuSeparator>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<shrtMenuSeparator />
```

# sourceEditor

This element provides the code view settings for use within Ephox EditLive!.

## Configuration Element Tree Structure

`<editLive>`  
`<sourceEditor>`

```
<editLive>
  ...
  <sourceEditor ... />
  ...
</editLive>
```

## Optional Attributes

### showBodyOnly

This attribute is a boolean and can only be *true* or *false*. If set to *true*, only the HTML between the body tags will be shown in Code View mode. HTML outside of the body (eg. style information) will be maintained.

Default Value: *false*

### enabled

If set to *true*, users will be able to access the Code View for documents. This attribute is a boolean and can only be *true* or *false*.

Default Value: *true*

## Example

The following example would ensure that only the content between the `<BODY>` tags was displayed.

```
<editLive>
  ...
  <sourceEditor showBodyOnly="true" />
  ...
</editLive>
```

The following example would allow users to view the source code of a document within EditLive!.

```
<editLive>
  ...
  <sourceEditor enabled="true" />
  ...
</editLive>
```

## Remarks

The `<sourceEditor>` element can appear only once within the `<editLive>` element.

If the `<sourceEditor>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<sourceEditor showBodyOnly="false" />
```

# spellCheck

# spellCheck (Applet)

This element defines the location of the JAR file to be used with the spell checker. This enables the dictionary to be defined for Tiny EditLive!.

If this element is omitted, the location of the spellcheck dictionary will be calculated by combining the [setDownloadDirectory Method](#) with the current user's locale.

**Example:** If the [DownloadDirectory](#) for an instance of EditLive! is specified as `http://yourserver:port/editlivejava` and the user's locale is Italian, the generated URL for spellcheck dictionary will be `http://yourserver:port/editlivejava/dictionaries/it_4_0.jar`.

## Configuration Element Tree Structure

<editLive>  
<spellCheck (Applet)>

```
<editLive>
  ...
  <spellCheck ... />
  ...
</editLive>
```

## Optional Attributes

A JAR file must be specified for the spell checking in the EditLive! Swing SDK to function. The name of the JAR file for the spell checker dictionary must be in **all** lower case letters. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where the EditLive! Swing SDK is being loaded from.

**jar**

The value for this attribute corresponds to the location of the JAR file to be used with EditLive! for spell checking. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where EditLive! is being loaded from.

The following attributes are booleans and can have the value of either *true* or *false*.

**startBackgroundChecking**

This attribute defines if spell check as you type is turned on or off as a default.

Default Value: *true*

**useNotModified**

This attribute defines whether the client running EditLive! will search the server hosting the EditLive! dictionaries for modified files. If this attribute is set to *true* when a modified dictionary is located on the server then this same file will be cached on the client.

Default Value: *true*

**startAutoCorrect**

This attribute defines whether the [Auto Correct](#) functionality is enabled on editor startup.

Default Value: *false*

## Example

The following example demonstrates how to define the location of the spell checker JAR file to be used with EditLive!. Spell check as you type is turned on as a default and spell checking dictionaries that have been modified are cached.

```
<editLive>
  ...
  <spellCheck
    jar="../../../redistributables/editlivejava/dictionaries/en_us_3_1.jar"
    startBackgroundChecking="true"
    useNotModified="false"
  />
  ...
</editLive>
```

## Remarks

The **<spellCheck>** element can appear only once within the **<editLive>** element.

If the **<spellCheck>** element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be self closing. It should appear as below:

```
<spellCheck jar=... />
```



# spellCheck (Swing SDK)

This element defines the location of the JAR file to be used with the spell checker. This enables the dictionary to be defined for Tiny EditLive!.

If this element is omitted, the location of the spellcheck dictionary will be calculated by combining the [setDownloadDirectory Method](#) with the current user's locale.

**Example:** If the [DownloadDirectory](#) for an instance of EditLive! is specified as <http://yourserver:port/editlivejava> and the user's locale is Italian, the generated URL for spellcheck dictionary will be [http://yourserver:port/editlivejava/dictionaries/it\\_4\\_0.jar](http://yourserver:port/editlivejava/dictionaries/it_4_0.jar).

## Configuration Element Tree Structure

<editLive>  
<spellCheck (Swing SDK)>

```
<editLive>
  ...
  <spellCheck ... />
  ...
</editLive>
```

## Required Attributes

A JAR file must be specified for the spell checking in the EditLive! Swing SDK to function. The name of the JAR file for the spell checker dictionary must be in **all** lower case letters. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where the EditLive! Swing SDK is being loaded from.

jar

The value for this attribute corresponds to the location of the JAR file to be used with EditLive! for spell checking. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where EditLive! is being loaded from.

## Optional Attributes

The following attributes are booleans and can have the value of either *true* or *false*.

startBackgroundChecking

This attribute defines if spell check as you type is turned on or off as a default.

Default Value: *true*

useNotModified

This attribute defines whether the client running EditLive! will search the server hosting the EditLive! dictionaries for modified files. If this attribute is set to *true* when a modified dictionary is located on the server then this same file will be cached on the client.

Default Value: *true*

startAutoCorrect

This attribute defines whether the [Auto Correct](#) functionality is enabled on editor startup.

Default Value: *false*

## Example

The following example demonstrates how to define the location of the spell checker JAR file to be used with EditLive!. Spell check as you type is turned on as a default and spell checking dictionaries that have been modified are cached.

```
<editLive>
  ...
  <spellCheck
    jar="../../../redistributables/editlivejava/dictionaries/en_us_3_1.jar"
    startBackgroundChecking="true"
    useNotModified="false"
  />
  ...
</editLive>
```

## Remarks

The `<spellCheck>` element can appear only once within the `<editLive>` element.

If the `<spellCheck>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<spellCheck jar=... />
```

# style

This element provides the information which is to be stored between the <STYLE> tags, between the <HEAD> tags of the Ephox EditLive! document. The element has no attributes.

## Configuration Element Tree Structure

```
<editLive>  
<document>  
<html>  
<head>  
<style>
```

```
<editLive>  
  <document>  
    <html>  
      <head>  
        <style>  
          <!--style configuration settings-->  
        </style>  
      </head>  
      ...  
    </html>  
  </document>  
  ...  
</editLive>
```

## Example

The following example demonstrates how to specify an embedded style sheet for use with EditLive!. The style sheet defined sets the font size to 14pt and the font to Arial for the <BODY> of the document.

```
<editLive>  
  <document>  
    <html>  
      <head>  
        <style>  
          p {font-size:14pt}  
          body {font-family:Arial}  
        </style>  
        ...  
      </head>  
      ...  
    </html>  
  </document>  
  ...  
</editLive>
```

## Remarks

Conflicts between externally linked style sheets and embedded style sheets in EditLive! are resolved according to the CSS precedence rules. Those rules state that any styles defined in embedded style sheets take precedence over those defined in an external style sheet. Thus, styles defined in the <style> element of the EditLive! configuration file have precedence over those defined in an external style sheet linked to via the <link> element within the configuration file.

The <style> element can appear only once within the <head> element.

# submenu

This element contains information for a submenu item which may be placed within a menu. The *Font*, *Font Size* and *Style* submenus are an example of this.

## Configuration Element Tree Structure

```
<editLive>
<menuBar>
<menu>
<submenu>
```

```
<editLive>
  ...
  <menuBar>
    <menu>
      <submenu ... />
    </menu>
  </menuBar>
  ...
</editLive>
```

## Required Attributes

name

This attribute specifies the name of the submenu. A list of available submenu items and their related name attribute can be found in the Submenu Items section of the [Menu and Toolbar Item List](#).

## Child Elements

```
<menuitem>
```

This element contains information for an item on the menu (eg. font items or font size items).

## Examples

The following example demonstrates how to include the Font submenu in the Format menu.

```
<editLive>
  ...
  <menuBar>
    <menu name="Format">
      <submenu name="FontFace">
        ...
      </submenu>
    </menu>
  </menuBar>
  ...
</editLive>
```

## Remarks

The `<submenu>` element can appear multiple times within the `<menu>` element.

# symbol

This element allows specifying a single symbol to appear in the insert symbol dialog. Any number of **<symbol>** elements, or none at all, can be nested in the parent **<symbols>** element.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<symbols>  
<symbol>
```

```
<editLive>  
  ...  
  <wysiwygEditor>  
    ...  
    <symbols>  
      <symbol />  
    </symbols>  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Required Attributes

char

This attribute defines the character to add to the insert symbol dialog. Characters are defined as a string.

## Example

The following example specifies how to remove the default symbols from the symbol insertion dialog, and add the symbols ^, ? and \*.

```
<editLive>  
  ...  
  <wysiwygEditor>  
    <symbols clearDialog="true">  
      <symbol char="^" />  
      <symbol char="?" />  
      <symbol char="*" />  
    </symbols>  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Remarks

The **<symbol>** element can appear any amount of times within the **<symbols>** element.

# symbols

This element allows the configuration of the default symbols to appear in the insert symbol dialog.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<symbols>
```

```
<editLive>  
  ...  
  <wysiwygEditor>  
    <symbols>  
      <!-- symbols settings -->  
    </symbols>  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Optional Attributes

`clearDialog`

This attribute defines whether all of the default symbols contained in the symbol insertion dialog will be removed. This attribute is a boolean and can have the value of either *true* or *false*.

Default Value: *false*

## Child Elements

`<symbol>`

This structure allows inserting a single symbol to appear in the symbol insertion dialog.

## Example

The following example would cause the symbol insertion dialog to not contain any symbols.

```
<editLive>  
  ...  
  <wysiwygEditor>  
    <symbols clearDialog="true">  
      ...  
    </symbols>  
  </wysiwygEditor>  
  ...  
</editLive>
```

## Remarks

The `<symbols>` element can appear only once within the `<wysiwygEditor>` element.

If the `<symbols>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. It should appear as below:

```
<symbols/>
```

# template

EditLive!'s [Template Browser](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

This element defines an individual template. The user can insert this template into EditLive! using the Template Browser plugin.

## Configuration Element Tree Structure

```
<editLive>
<templates>
<category>
<template>
```

```
<editLive>
  ...
  <templates>
    <category name="Category 1">
      <template ... />
    </category>
  </templates>
  ...
</editLive>
```

## Required Attributes

name

The name of the template that will be displayed in the Template Browser.

value

This is where the template that will be inserted into EditLive! by the user is defined. The template may contain HTML, but it must be [URL encoded](#).

### Example

`<div class="firstClass">text here</div >` would be encoded as: `value=%3Cdiv+class%3D%22firstClass%22%3Etext+here%3C%2Fdiv%3E`.

## Examples

The following example demonstrates how to define a heirarchy of templates.

```
<editLive>
  ...
  <templates>
    <category name="Category 1">
      <category name="Sub Category 1">
        <template name="Template 1" value="%3Cp%3EFirst+template%3C%2Fp%3E">
        </category>
      <category name="Sub Category 2">
        <template name="Template 2" value="%3Cdiv%3ESecond+template%3C%2Fdiv%3E">
        </category>
      ...
    </category>
    ...
  </templates>
  ...
</editLive>
```

## Remarks

The `<template>` element can appear multiple times within the `<category>` element.

The `<template>` element must be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<template name=... />
```

Text assigned to the value attribute must be [URL encoded](#) as it is in the example above.

## See Also

- [Encoding Content for Use with EditLive!](#)



# templates

EditLive!'s [Template Browser](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

## Description

This element allows you to specify templates to be loaded into EditLive!'s Template Browser.

## Configuration Element Tree Structure

<[editLive](#)>  
<[templates](#)>

```
<editLive>
  ...
  <templates>
    ...
  </templates>
  ...
</editLive>
```

## Child Elements

<[category](#)>

This element specifies a category into which one or many templates can be added.

<[template](#)>

This element specifies a template that will appear in the Template Browser.

# textImport

This element configures the manner in which EditLive! reacts when a user attempts to paste plain text into the editor. This includes plain text copied from locations such as Microsoft Notepad.

## Configuration Element Tree Structure

```
<editLive>  
<textImport>
```

```
<editLive>  
  ...  
  <textImport ... />  
  ...  
</editLive>
```

## Optional Attributes

### linebreakCreatesBR

This boolean value specifies whether pasted newline characters are preserved as `<br />` tags or whether paragraphs are created instead.

Default value: *false*

### oneLineBreakPerParagraph

This boolean value specifies the behaviour for copying content from EditLive! into plain text formats. Setting this attribute to *true* will append a single newline character after each paragraph found. Setting this attribute to *false* will append two newline characters after each paragraph found.

Default value: *false*

## Example

The following example demonstrates how to set EditLive! to preserve pasted newline characters from plain text as `<br />` tags.

```
<editLive>  
  ...  
  <textImport linebreakCreatesBR="true" />  
  ...  
</editLive>
```

## Remarks

The `<textImport>` element can appear only once within the `<editLive>` element.

If the `<textImport>` element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<textImport linebreakCreatesBR=.../>
```

**thesaurus**

# thesaurus (Applet)

This element defines the location of the JAR file to be used with the thesaurus. This enables the dictionary to be defined for EditLive!.

If this element is omitted, the location of the thesaurus will be calculated by combining the [setDownloadDirectory Method](#) with the current user's locale.



**Example:** If the [DownloadDirectory](#) for an instance of EditLive! is specified as `http://yourserver:port/editlivejava` and the user's locale is Canada, the generated URL for thesaurus will be `http://yourserver:port/editlivejava/thesaurus/thes_ca_6_0.jar`.

## Configuration Element Tree Structure

<editLive>  
<thesaurus (Applet)>

```
<editLive>
  ...
  <thesaurus ... />
  ...
</editLive>
```

## Optional Attributes

jar

The value for this attribute corresponds to the location of the JAR file to be used with EditLive! for the thesaurus. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where EditLive! is being loaded from.

useNotModified

This attribute defines whether the client running EditLive! will search the server hosting the EditLive! thesaurus for modified files. This attribute is a boolean and can have the value of either `true` or `false`. If this attribute is set to `true` when a modified thesaurus is located on the server this same file will be cached on the client.

Default Value: `true`

*useNotModified was removed in EditLive 9.0.2. The thesaurus is checked against the server per standard HTTP caching mechanisms.*



*If you wish the thesaurus to be cached for a long time, configure your HTTP server to set the "expires" header for the thesaurus for some time in the future.*

## Example

The following example demonstrates how to define the location of the spell checker JAR file to be used with EditLive!. Thesauruses that have been modified are cached.

```
<editLive>
  ...
  <thesaurus
    jar="../../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar"
    useNotModified="false"
  />
  ...
</editLive>
```

## Remarks

The **<thesaurus>** element can appear only once within the **<editLive>** element.

If the **<thesaurus>** element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<thesaurus jar=... />
```

# thesaurus (Swing SDK)

This element defines the location of the JAR file to be used with the thesaurus. This enables the dictionary to be defined for EditLive!.

If this element is omitted, the location of the thesaurus will be calculated by combining the [setDownloadDirectory Method](#) with the current user's locale.



**Example:** If the [DownloadDirectory](#) for an instance of EditLive! is specified as <http://yourserver:port/editlivejava> and the user's locale is Canada, the generated URL for thesaurus will be [http://yourserver:port/editlivejava/thesaurus/thes\\_ca\\_6\\_0.jar](http://yourserver:port/editlivejava/thesaurus/thes_ca_6_0.jar).

## Configuration Element Tree Structure

<editLive>  
<thesaurus (Swing SDK)>

```
<editLive>
  ...
  <thesaurus ... />
  ...
</editLive>
```

## Required Attributes

A JAR file must be specified for the thesaurus in the EditLive! Swing SDK to function. The name of the thesaurus JAR file must be in all lowercase letters.

jar

The value for this attribute corresponds to the location of the JAR file to be used with EditLive! for the thesaurus. The URL location of the JAR file can either be an absolute URL or a URL relative to the page where EditLive! is being loaded from.

## Optional Attributes

useNotModified

This attribute defines whether the client running EditLive! will search the server hosting the EditLive! thesaurus for modified files. This attribute is a boolean and can have the value of either *true* or *false*. If this attribute is set to *true* when a modified thesaurus is located on the server this same file will be cached on the client.

Default Value: *true*

## Example

The following example demonstrates how to define the location of the spell checker JAR file to be used with EditLive!. Thesauruses that have been modified are cached.

```
<editLive>
  ...
  <thesaurus
    jar="../../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar"
    useNotModified="false"
  />
  ...
</editLive>
```

## Remarks

The <thesaurus> element can appear only once within the <editLive> element.

If the <thesaurus> element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<thesaurus jar=... />
```

# title

This element provides the information which is to be stored between the <TITLE> tags, between the <HEAD> tags of the EditLive! document. The element has no attributes.

## Configuration Element Tree Structure

```
<editLive>
<document>
<html>
<head>
<title>
```

```
<editLive>
  <document>
    <html>
      <head>
        <title>
          <!--title configuration settings-->
        </title>
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Example

The following example demonstrates how to specify the title of the EditLive! document to be *This is the Title*.

```
<editLive>
  <document>
    <html>
      <head>
        <title>This is the Title</title>
        ...
      </head>
      ...
    </html>
  </document>
  ...
</editLive>
```

## Remarks

The <title> element can appear only once within the <head> element.

# toolbar

This element contains the configuration information for a toolbar for use within EditLive!.

Items will appear in a toolbar in EditLive!, from left to right, in the order that they appear in the EditLive! configuration document.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
```

```
<editLive>
  ...
  <toolbars>
    <toolbar ...>
      <!--toolbar configuration settings-->
    </toolbar>
  </toolbars>
  ...
</editLive>
```

## Required Attributes

name

An identifying name for this toolbar. The value for this attribute must be unique within the EditLive! collection of toolbars.

## Child Elements

<toolbarButton>

This element will cause a particular button to be present on the toolbar within EditLive!.

<toolbarButtonGroup>

This element will cause a particular group of buttons to be present on the toolbar within EditLive!. The operation of these buttons within EditLive! will be mutually exclusive. The buttons added by this element can only be added and removed from the toolbar as a group.

For example, the alignment buttons are a button group as Align Left cannot be activated at the same time as Align Right.

<toolbarComboBox>

This element will cause a particular combo box to be present on the toolbar within EditLive!.

<toolbarSeparator>

This element will cause the appearance of a vertical separating line between toolbar elements.

<customToolbarButton>

This element specifies the properties for a developer defined custom toolbar button for use within EditLive!.

<customToolbarComboBox>

This element specifies the properties for a developer defined custom toolbar combo box for use within EditLive!.

## Example

This example demonstrates how to declare toolbars with the names *command* and *format*.

```
<editLive>
  ...
  <toolbars>
    ...
    <toolbar name="command">
      ...
    </toolbar>
    <toolbar name="format">
      ...
  </toolbars>
</editLive>
```

```
    </toolbar>
    ...
  </toolbars>
  ...
</editLive>
```

## Remarks

Toolbars will appear in the EditLive! interface in the order that they appear in the configuration file.

The `<toolbar>` element can appear multiple times within the `<toolbars>` element.



# toolbarButton

This element will cause a particular button to be present on the toolbar within EditLive!.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<toolbarButton>
```

```
<editLive>
  ...
  <toolbars>
    <toolbar>
      <toolbarButton ... />
    </toolbar>
  </toolbars>
  ...
</editLive>
```

## Required Attributes

name

This attribute gives the name for the button. A list of possible toolbar button items and their associated name attributes can be found in the [Menu and Toolbar Item List](#).

## Example

The following example demonstrates how to add the **Cut**, **Copy**, and **Paste** buttons to the Command Toolbar.

```
<editLive>
  ...
  <toolbars>
    <toolbar name="command">
      <toolbarButton name="Cut" />
      <toolbarButton name="Copy" />
      <toolbarButton name="Paste" />
    </toolbar>
  </toolbars>
  ...
</editLive>
```

## Remarks

The `<toolbarButton>` element can appear multiple times within the `<toolbar>` elements.

The `<toolbarButton>` element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<toolbarButton name=... />
```

# toolbarButtonGroup

This element will cause a particular group of buttons to be present on the toolbar within EditLive!. The buttons added by this element can only be added and removed from the toolbar as a group.

The operation of these buttons within EditLive! will be mutually exclusive. For example, the alignment buttons are a button group as Align Left cannot be activated at the same time as Align Right.

## Configuration Element Tree Structure

```
<editLive>  
<toolbars>  
<toolbar>  
<toolbarButtonGroup>
```

```
<editLive>  
  ...  
  <toolbars>  
    <toolbar>  
      <toolbarButtonGroup ... />  
    </toolbar>  
  </toolbars>  
  ...  
</editLive>
```

## Required Attributes

name

This attribute gives the name for the button group. The [Menu and Toolbar Item List](#) details the available groups and the associated name attribute.

## Example

The following example demonstrates how to add the alignment buttons to the Format Toolbar.

```
<editLive>  
  ...  
  <toolbars>  
    <toolbar name="format">  
      <toolbarButtonGroup name="Align" />  
    </toolbar>  
  </toolbars>  
  ...  
</editLive>
```

## Remarks

The **<toolbarButtonGroup>** element can appear multiple times within the **<toolbar>** elements.

The **<toolbarButtonGroup>** element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<toolbarButtonGroup name=... />
```

# toolbarComboBox

This element will cause a particular combo box to be present on the toolbar within EditLive!

## Configuration Element Tree Structure

```
<editLive>  
<toolbars>  
<toolbar>  
<toolbarComboBox>
```

```
<editLive>  
  ...  
  <toolbars>  
    <toolbar>  
      <toolbarComboBox ... >  
        <!--toolbar combo box configuration settings-->  
      </toolbarComboBox>  
    </toolbar>  
  </toolbars>  
  ...  
</editLive>
```

## Required Attributes

name

This attribute gives the name for the toolbar combobox item. A list of possible item names can be found in the [Menu and Toolbar Item List](#).

## Child Elements

```
<comboBoxItem>
```

This element contains the information required by EditLive! to configure an item within one of the EditLive! combo boxes.

## Example

The following example demonstrates how to add the Style combo box to the Format Toolbar.

```
<editLive>  
  ...  
  <toolbars>  
    <toolbar name="format">  
      <toolbarComboBox name="Style">  
        ...  
      </toolbarComboBox>  
    </toolbar>  
  </toolbars>  
  ...  
</editLive>
```

## Remarks

The `<toolbarComboBox>` element can appear multiple times within the `<toolbar>` element.

# toolbars

This element contains the configuration information for the toolbars within EditLive!. This includes the Format and the Command toolbars and the buttons and combo boxes contained within them.

## Configuration Element Tree Structure

<editLive>  
<toolbars>

```
<editLive>
  ...
  <toolbars>
    <!--toolbars configuration settings-->
  </toolbars>
  ...
</editLive>
```

## Optional Attributes

### display

This attribute defines where the toolbars are placed in relation to the text area. This attribute has two possible values:

- *docked* - The toolbars appear above and separate to the text area
- *floating* - The toolbars appear floating to the top of the text area. This is usually used creating a minimal editing UI with [In Place Editing](#)

Default value: *docked*

## Child Elements

<toolbar>

This element contains the configuration information for a toolbar within EditLive!.

## Remarks

The <toolbars> element can appear only once within the <editLive> element.

# toolbarSeparator

This element will cause the appearance of a vertical separating line between toolbar elements in Tiny EditLive!.

This element has no attributes or child elements.

## Configuration Element Tree Structure

```
<editLive>
<toolbars>
<toolbar>
<toolbarSeparator>
```

```
<editLive>
...
<toolbars>
  <toolbar>
    <toolbarSeparator />
  </toolbar>
</toolbars>
...
</editLive>
```

## Example

The following example demonstrates how to insert a toolbar separator between the Paste and Find buttons on the Command Toolbar.

```
<editLive>
...
<toolbars>
  <toolbar name="command">
    <toolbarButton name="tlbCut" />
    <toolbarButton name="tlbCopy" />
    <toolbarButton name="tlbPaste" />
    <toolbarSeparator />
    <toolbarButton name="tlbFind" />
  </toolbar>
</toolbars>
...
</editLive>
```

## Remarks

The **<toolbarSeparator>** element can appear multiple times within the **<toolbar>** element.

The **<toolbarSeparator>** element must be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. See the example below:

```
<toolbarSeparator />
```

# trackChanges

## Description

This element defines which colors are used for remove or change operations performed when using the [Track Changes](#) functionality of EditLive!. The color specified for either of these operations will appear the same for every EditLive! user.

## Configuration Element Tree Structure

```
<editLive>  
<wysiwygEditor>  
<trackChanges>
```

```
<editLive>  
...  
<wysiwygEditor>  
  <trackChanges />  
</wysiwygEditor/>  
...  
</editLive>
```

## Optional Attributes

### remove

The color used by EditLive! to represent any *remove* operations performed in the editor when using [Track Changes](#). The value specified by this attribute needs to be the hexadecimal representation for the desired color (e.g. `remove="#FF0000"`).

### change

The color used by EditLive! to represent any *change* operations performed in the editor when using [Track Changes](#). The value specified by this attribute needs to be the hexadecimal representation for the desired color (e.g. `change="#FF0000"`).

## Example

The following example demonstrates how to configure [Track Changes](#) to render all *change* operations by all users as blue (represented by #00EEEE) and render all *remove* operations by all users as red (represented by #FF0000).

```
<editLive>  
...  
<wysiwygEditor>  
  <trackChanges remove="#FF0000" change="#00EEEE" />  
</wysiwygEditor>  
...  
</editLive>
```

# type

This element defines the properties of an multimedia (object) type which is to be used with EditLive!. Some of the attributes of this element map to the attributes of the <OBJECT> element which is inserted into EditLive! when a multimedia type is inserted.

## Configuration Element Tree Structure

```
<editLive>
<mediaSettings>
<multimedia>
<types>
<type>
```

```
<editLive>
...
<mediaSettings>
...
<multimedia>
...
<types>
...
<type>
<!--type configuration settings-->
</type>
...
</types>
</multimedia>
</mediaSettings>
...
</editLive>
```

## Required Attributes

name

A unique name for the multimedia type. This value is displayed to the user as a option in the multimedia object dialog in the list of available types.

## Optional Attributes

allowCustomParams

This boolean attribute specifies if users can provide custom defined parameters in addition to those configured via the <param> elements in the configuration file. This attribute is a boolean with possible values of *true* or *false*. When set to *true*, users can specify custom parameters.

Default Value: *true*

extension

The file extension to associate with this object type.

urlParam

This attribute specifies the name of the parameter which is to be used to specify the location (URL) for the object's source files.

classid

The meaning of this attribute is browser dependent. Please see information from the provider of your browser(s).

codebase

The meaning of this attribute is browser dependent. Please see information from the provider of your browser(s). In some browsers this attribute is used to specify the download location for the plug-in or program used to render the object.

type

The meaning of this attribute is browser dependent. Please see information from the provider of your browser(s). In some browsers this attribute is used to specify the MIME type for the object.

## Child Elements

`<param>`

This element enables `<param>` tags associated with the `<object>` tag to be inserted into the code in EditLive!.

## Example

The following example demonstrates how to configure a multimedia type for use with EditLive!. In this case the media type is a Windows Media Player streaming type with the .asx extension.

```
<editLive>
...
<mediaSettings>
...
  <multimedia>
    <types>
      ...
      <type
        name="Windows Media (Streaming)"
        type="application/x-mplayer2"
        extension="asx"
        allowCustomParams="true"
        urlParam="fileName"
      >
        <param name="animationAtStart" />
        <param name="autoStart" />
        <param name="clickToPlay" />
      </type>
      ...
    </types>
  </multimedia>
</mediaSettings>
...
</editLive>
```

## Remarks

The `<type>` element can appear multiple times within the `<types>` element.



# types

This element allows for the provision of a list of multimedia file types to be supported by Tiny EditLive!.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<multimedia>  
<types>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    ...  
    <multimedia>  
      <types>  
        <!--types configuration settings-->  
      </types>  
    </multimedia>  
  </mediaSettings>  
  ...  
</editLive>
```

## Child Elements

```
<type>
```

This element allows the configuration of a specific multimedia file type.

## Remarks

The `<types>` element can appear only once within the `<multimedia>` element.

# webdav

WebDAV support has been removed in EditLive! 9.1

This element contains the configuration information for the use of EditLive! with WebDAV repositories.

## Configuration Element Tree Structure

When the <webdav> element appears within the <multimedia> configuration element, information contained within the <webdav> element is used to configure the WebDAV support for embedded objects and multimedia file types in EditLive!.

<editLive>  
<mediaSettings>  
<multimedia>  
<webdav>

```
<editLive>
...
<mediaSettings>
  <multimedia>
    ...
    <webdav>
      <!--webdav configuration settings-->
    </webdav>
  </multimedia>
</mediaSettings>
...
</editLive>
```

When the <webdav> element appears within the <images> element, the configuration information contained within the <webdav> element is used to configure the WebDAV support for image repositories in EditLive!.

<editLive>  
<mediaSettings>  
<images>  
<webdav>

```
<editLive>
...
<mediaSettings>
  <images>
    ...
    <webdav>
      <!--webdav configuration settings-->
    </webdav>
  </images>
</mediaSettings>
...
</editLive>
```

When the <webdav> element appears within the <hyperlinks> element, the configuration information contained within the <webdav> element is used to configure the WebDAV support for hyperlinks in EditLive!.

<editLive>  
<hyperlinks>  
<webdav>

```
<editLive>
...
<hyperlinks>
  ...
  <webdav>
    <!--webdav configuration settings-->
  </webdav>
  ...
</hyperlinks>
...
</editLive>
```

---

## Child Elements

[<repository>](#)

This element defines the location of a WebDAV repository.

## Remarks

The `<webdav>` element can appear only *once* within the [<images>](#) element.

The `<webdav>` element can appear only *once* within the [<hyperlinks>](#) element.

The `<webdav>` element can appear only *once* within the [<multimedia>](#) element.

To use the WebDAV functionality of EditLive!, the UseWebDAV load-time property must be set to *true*.

# webeqLicense

This element allows for specifying licenses for Design Science MathML related products. This element is designed for OEM vendors incorporating EditLive! into their system with [Equation Editing](#) functionality.

## Configuration Element Tree Structure

[<editLive>](#)  
[<otherLicenses>](#)  
[<mathml>](#)

```
<editLive>
  ...
  <otherLicenses>
    <webeqLicense ... />
  </otherLicenses>
  ...
</editLive>
```

## Optional Attributes

[key](#)

This attribute specifies the key for the Style Editor packaged with the Design Science MathFlow product.

[equationComposerKey](#)

This attribute specifies the key for the Equation Composer packaged with the Design Science MathFlow product.

## See Also

- [<mathml>](#) Configuration File Element

# wordImport

This element configures the manner in which Tiny EditLive! reacts when text is imported from Microsoft Word.

## Configuration Element Tree Structure

<editLive>  
<wordImport>

```
<editLive>
  ...
  <wordImport ... />
  ...
</editLive>
```

## Optional Attributes

### styleOption

This attribute specifies the user prompting and behaviour of EditLive! upon detecting an import from Microsoft Word. This attribute has five possible values:

- *user\_prompt* - Users will be prompted as to whether they wish to import Microsoft Word styles as inline or embedded styles or strip them from the imported text. The dialog used to prompt the user is the same as the Paste Special dialog.
- *merge\_embedded\_styles* - This setting will result in EditLive! importing styles from Microsoft Word with the styles stored in the <head> of the document and class attributes will be applied where relevant.
- *merge\_inline\_styles* - This setting will result in EditLive! importing styles from Microsoft Word with style information applied inline in place of class attributes.
- *clean* - The clean setting will result in EditLive! stripping the Microsoft Word Styles from the imported text. Only structural HTML elements will be imported, such as <b>, <p>, etc.
- *plain\_text* - The plain\_text setting will strip out all style information from the imported text.

Styles imported from Microsoft Word will not overwrite styles which already exist within the document.

### cleanOption

This boolean value specifies whether the *Clean HTML* option will appear in the **Paste Special...** dialog.

Default value: *true*

### mergeInlineStylesOption

This boolean value specifies whether the *Styled HTML (Inline)* option will appear in the **Paste Special...** dialog.

Default value: *true*

### mergeEmbeddedStylesOption

This boolean value specifies whether the *Styled HTML (Embedded)* option will appear in the **Paste Special...** dialog.

Default value: *true*

### plainTextOption

This boolean value specifies whether the *Plain Text* option will appear in the **Paste Special...** dialog.

Default value: *true*

## Example

The following example demonstrates how to set EditLive! to prompt the user with style import options every time a Microsoft Word import is detected.

```
<editLive>
  ...
  <wordImport styleOption="user_prompt" />
  ...
</editLive>
```

## Remarks

The **<wordImport>** element can appear only once within the **<editLive>** element.

If the **<wordImport>** element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<wordImport styleOption=.../>
```

# wysiwygEditor

This element allows you to set options relating to the EditLive! editing pane.

## Configuration Element Tree Structure

<editLive>  
<wysiwygEditor>

```
<editLive>
...
<wysiwygEditor>
  <!--wysiwygEditor settings-->
<wysiwygEditor/>
...
</editLive>
```

## Optional Attributes

### tabPlacement

This attribute defines where the Design and Code view tabs are placed on the editor pane. This attribute has three possible values:

- *top* - Places the tabs at the top of the editor pane.  
*Top* tab placement is only available in the Swing SDK. If specified in the applet, it will have no effect.
- *bottom* - Places the tabs at the bottom of the editor pane.
- *off* - Removes the tabs from the editor.

Default Value: *top*

**The following *optional* attributes are booleans and can have the value of either *true* or *false*.**

### showDocumentNavigator

This boolean attribute defines whether the Document Navigator toolbar is displayed in the interface. The document navigator displays the current location in the HTML structure to the user and enables them to easily select any element in the current path and the contents of the element.

Default Value: *true*

### brOnEnter

This boolean attribute defines whether a <br> or <p> tag is inserted when the Enter key is pressed. When set to *true* a <br> is inserted when Enter is pressed; a <p> tag is inserted when the Shift + Enter key combination is used. When set to *false* this behavior is reversed.

Pressing enter when in table fields will always yield a <br> tag, regardless of the value set to *brOnEnter*.

The default behavior for EditLive! is to insert a <p> on Enter and a <br> on Shift + Enter. This is the same behavior as when this setting is set to *false*.

Default Value: *false*

### enableTrackChanges

This boolean attribute defines if track changes is enabled or disabled when the editor loads.

If a HTML document is loaded into EditLive! that already contains change information, track changes will always turn on automatically.

Default Value: *false*

### disableInlineImageResizing

This boolean attribute defines whether users will be able to manually resize images placed into an instance of EditLive! through the design view.

Default Value: *false*

### disableInlineTableResizing

This boolean attribute defines whether users will be able to manually resize tables placed into an instance of EditLive! through the design view.

Default Value: *false*

### openExternalLinks

This boolean attribute defines whether users can open external links from within EditLive!. If true, users can open external links by ctrl-clicking on PC, or option-clicking on Mac.

Default Value: *true*

This attribute has no effect on internal links e.g.



```
<a href="#top">top</a>
```

### shiftSpaceInsertsNBSP

This boolean attribute defines the behaviour of the Shift+Space key combination. When set to *true* this combination will insert a non-breaking space character. When set to *false* it will insert a normal white space character.

Default Value: *false*

### showGridlines

This boolean attribute defines whether table and section gridlines are shown by default. When set to *false*, the "Show Gridlines" button can still be used to show them.

Default Value: *true*

### showSectionGridlines

This boolean attribute defines whether gridlines are shown around sections (HTML DIV tags). When set to *false*, gridlines are never shown on sections.

The **showSectionGridlines** setting is ignored if **showGridlines** is false - this setting allows you to turn off Section gridlines when **showGridlines** is set to true.

Default Value: *true*

### useEphoxLookAndFeel

This boolean attribute specifies whether the Tiny look and feel is used for the editor. If set to *false*, the system look and feel is used.

Default Value: *true*

### useNameInBookmarks

This boolean attribute specifies whether to use the name attribute in bookmarks. If set to *true*, both the name and id attributes will be added to bookmarks; otherwise, only the id attribute will be added.

Default value: *false*

If both *useNameInBookmarks* and the *xhtmlStrict* option in `<htmlFilter>` are set to be *true*, *useNameInBookmarks* will override *xhtmlStrict* and continue to insert the name attribute in bookmarks. This will make your document non-XHTML Strict compliant.

### allowLocalImagesWithoutUploader

This boolean attribute specifies whether to display local images and allow local image insertion if no image upload handler is defined.

When set to false, and no image uploader is set, two things happen:

- local images in the editor will display as broken images (eg pasted from MS word)
- the insert image dialog will hide the local image option
  - This is the equivalent to doing `<mediaSettings><images allowLocalImages="false">` in the config, but allows it to dynamically change based on whether or not an image uploader is set

Default value: *true*

If the only uploader that is set comes from a background-loading plugin that uses `ELJBean.setFileUploader()` (eg connections), and the plugin takes a while to download, local images in the content will display as broken and then refresh to the correct image once the plugin has set a file uploader.

### stylesVisibility

This attribute defines how to treat CSS styles with the attribute *ephox-visible* when determining what to include in the Styles drop down in EditLive!. For more information see [Restricting what CSS classes appear in EditLive! styles drop down](#)

This attribute has two possible values:

- *blacklist* - only CSS Styles where *ephox-visible* is **not** "false" will be included.
- *whitelist* - only CSS Styles where *ephox-visible* is "true" will be included.

Default Value: *blacklist*



## Child Elements

### <customTags>

This structure allows for the configuration of settings related to EditLive!'s support of custom tags (both registered and unknown).

### <symbols>

This structure allows for the configuration of the settings related to the symbol insertion dialog.

### <colorPalette>

The color palette allows developers to specify the colors displayed in EditLive!'s color selection dialog (as seen when selecting text foreground or background color).

### <trackChanges>

Customize insertion and deletion rendering for all users of the [Track Changes](#) functionality.

## Example

The following example would display the Design and Code tabs on the bottom of the editor pane.

```
<editLive>
  ...
  <wysiwygEditor tabPlacement="bottom" />
  ...
</editLive>
```

## Remarks

The **<wysiwygEditor>** element can appear only once within the **<editLive>** element.

If the **<wysiwygEditor>** element is to be left blank the element must then be a complete tag; it cannot contain a tag body. Therefore the tag must be closed in the same line. It should appear as below:

```
<wysiwygEditor tabPlacement=.../>
```

# services

This element allows for the provisioning of a list of oEmbed services to be supported by Tiny EditLive!.

## Configuration Element Tree Structure

[<editLive>](#)  
[<mediaSettings>](#)  
[<multimedia>](#)  
[<services>](#)

```
<editLive>
  ...
  <mediaSettings>
    ...
    <multimedia>
      <services>
        <!--service configuration settings-->
      </services>
    </multimedia>
  </mediaSettings>
  ...
</editLive>
```

## Child Elements

[<service>](#)

This element allows the configuration of a specific multimedia file type.

## Remarks

The [<services>](#) element can appear only once within the [<multimedia>](#) element.

# service

This element defines the properties of an oEmbed service which is to be used with EditLive!. These attributes map to the configuration according to the [oEmbed specification](#).

EditLive! evaluates service elements in the order they appear within the configuration file, allowing for fallback providers to be used.

## Configuration Element Tree Structure

```
<editLive>  
<mediaSettings>  
<multimedia>  
<services>  
<service>
```

```
<editLive>  
  ...  
  <mediaSettings>  
    ...  
    <multimedia>  
      ...  
      <services>  
        ...  
        <!--service configuration settings-->  
        <service />  
      ...  
    </services>  
  </multimedia>  
</mediaSettings>  
  ...  
</editLive>
```

## Required Attributes

name

The name of the oEmbed service.

endpoint

The API endpoint of the oEmbed service. EditLive! supports both XML and JSON API endpoints.

scheme

The scheme to match URLs against that should use this endpoint. Wildcards are supported.

## Example

The following example demonstrates how to configure multiple oEmbed services for use with EditLive!. In this case the first service is YouTube, with a fallback to embed.ly for all other URLs.

```
<editLive>  
  ...  
  <mediaSettings>  
    ...  
    <multimedia>  
      <services>  
        ...  
        <service name="YouTube" endpoint="http://www.youtube.com/oembed" scheme="http://*.youtube.com/" />  
        <service name="Embed.ly" endpoint="http://api.embed.ly/1/oembed?key=<your key here>" scheme="*" />  
      />  
    ...  
  </services>  
</multimedia>  
</mediaSettings>  
  ...  
</editLive>
```

#### Note



Some endpoints (such as embed.ly) require API keys specific to your application, and often the free account limits are very low (for example embed.ly only supports 5000 requests a month on free accounts).

API keys can be specified along with the endpoint URL, as shown in the example above.

#### Remarks

The `<service>` element can appear multiple times within the `<services>` element.

# contentLanguages

This element allows for the definition of a list of languages to be made available to the end users of EditLive! for application via the Content Language tool. The Content Language tool enables users to mark up a document to indicate which language the content is in through the application of the HTML *lan* attribute.

## Configuration File Element

`<editLive>`  
`<contentLanguages>`

```
<editLive>
  ...
  <contentLanguages>
    <!--language configuration settings-->
  </contentLanguages>
  ...
</editLive>
```

## Child Elements

`<language>`

This element allows for the specification of languages to be shown as options for the Content Language tool.

## Remarks

The `<contentLanguages>` element can appear only once within the `<editLive>` element.

# language

This element defines a language. The user is able to apply this language to content in EditLive! via the Content Language tool. The Content Language tool enables users to mark up a document to indicate which language the content is in through the application of the HTML *lang* attribute.

## Configuration File Element

```
<editLive>  
<contentLanguages>  
<language>
```

```
<editLive>  
  ...  
  <contentLanguages>  
    <language ...>  
  </contentLanguages>  
  ...  
</editLive>
```

## Required Attributes

### lang

This attribute defines the language code that will be applied to the content.

## Optional Attributes

### display

This attribute defines the text that will be displayed on the dropdown for the Content Language tool. If unspecified and the language code is known by EditLive!, EditLive! will provide a language name that is translated in the language of the user interface.

### dir

This attribute defines the directionality of the content. If unspecified, the value will default to "ltr".

## Remarks

The <language> element can appear multiple times within the <contentLanguages> element.

The <language> element must be a complete tag; it cannot contain a tag body. Therefore, the tag must be closed in the same line. See the example below:

```
<language lang=... display=... dir=.../>
```

# Plugin XML Elements

EditLive!'s [Advanced API](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

Plugin XML is used to specify additional editor functionality to be loaded into EditLive!. Plugin XML is used to specify [Advanced API](#) invocations of EditLive! or additional Javascript files that can be incorporated into the webpage hosting EditLive!. Developers can also use the Plugin XML to specify additional menu items to allow these new features to be activated.

Because server-side languages can be used to generate documents of any types at run-time, Plugin XML files can also be stored as server-side language files (e.g. JSP files, ASP files), as long as the file contains calls to render the information as XML at run-time.

The following methods are used to specify Plugin XML:

- [addPlugin Method](#) (applet only)
- [addPluginAsText Method](#) (applet only)
- [<Plugins>](#) Configuration File Element

# plugin

This element defines the information required to load a single plugin.

## Plug-in Element Tree Structure

<plugin>

```
<plugin>
  <!-- configuration settings -->
</plugin>
```

## Child Elements

<advancedapis (Applet)>

This element defines what Java classes the plugin will load.

<menu>

The menu element defines the menu items which will appear under the *Plugins* menu in EditLive!.

<script>

Additional Javascript files can be applied to the webpage loading EditLive! by using this element.

## Optional Attributes

load

This attribute defines how java classes (defined in <advancedapis (Applet)>) and Javascript files (defined in <script>) are loaded by EditLive!:

- *immediate*- EditLive! will not load until each specified java class/classes and/or Javascript file has been cached and is ready for use by the editor.
  - **Example:** Your plugin creates custom rendering for custom HTML tags appearing in your content. Because these tags may appear in the default content for EditLive!, the plugin is required as soon as the editor loads. To ensure this plugin code is instantly available to EditLive!, use the immediate load attribute.
- *background*- EditLive! will load as normal, caching the required java class/classes in the background.
  - **Example:** Your plugin may be designed to launch a complex dialog for restructuring the current HTML stored in EditLive!. Due to its complexity, the plugin size is quite large and hence will take a significant amount of time to cache and load into EditLive!. It would be undesirable to use *immediate* because this would substantially increase the load time for EditLive!. *background* can be used to ensure the editor loads as normal, providing the plugin functionality to the user a short time after the editor has loaded.
- *lazy*- These java classes will not be cached and utilized by the editor until the user selects a menu item from the Plugins menu that requires these java classes.
  - **Example:** Your plugin defines a menu item that, once selected, inserts a HTML fragment into EditLive!. The plugin size is quite small, meaning it will not take long to both cache and load. The *lazyload* value could be used to only load the plugin when the user tries to utilize this functionality.

Only use *lazy* when implementing java classes you want loaded when the user selects a specified menu item. If you want the functionality found in your specified java classes or Javascript files available immediately, use *immediate*.

Default Value: *immediate*

url

This attribute is used when a plugin is loaded into the editor via the <Plugins> Configuration File element.

The url attribute is used to specify the location of an external XML file that specifies the details of an EditLive! plugin.

### Example

The following file myPlugin.xml sits on a server at the URL <http://myserver/mypluginsdir/myPlugin.xml>

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="immediate">
  <advancedapis jar="myPlugin.jar" class="MyPlugin" />
  <menu>
    <customMenuItem name="plugin" action="raiseEvent" value="displayDialog" text="Display Dialog"
  imageURL="images/small_logo.gif" />
  </menu>
</plugin>
```



This plugin can be loaded using a **<plugin>** element, with a url attribute, nested in the Configuration File's **<Plugins>** element:

```
<editLive>
    ...
    <plugins>
        <plugin url="http://myserver/mypluginsdir/myPlugin.xml" />
    </plugins>
    ...
</editLive>
```

## name

EditLive! comes packaged with several plugins. These plugins can be initialized by simply specifying a name attribute on a plugin element resident in an EditLive! configuration file.

EditLive! for Java accepts the following plugin *name* attributes:

- *imageEditor* - enables the [Image Editor](#) functionality.
- *tableToolbar* - allows developers to specify an **<inlineToolbar>** to appear for TABLE elements.
- *accessibility* - enables the [Accessibility As You Type](#) functionality.
- *BrokenHyperlinkReport* - enables the [Broken Hyperlink Report](#) functionality.
- *autosave* - enables the [Autosave](#) functionality.
- *rtfpaste* - provides the ability to import .rtf when using Apple OSX. The RTF clipboard flavor on OSX will be supported, allowing copy and paste of rich content from sources such as Safari.
- *spelling* - enables the [Spell Check](#) functionality.

## Example

The following configuration file excerpt will load the image editor and the inline table toolbar functionality into an instance of EditLive!.

```
<editlive>
    ...
    <plugins>
        <plugin name="imageEditor" />
        <plugin name="tableToolbar" />
    </plugins>
</editlive>
```

## Remarks

Multiple **<plugin>** elements can be applied to one plug-in XML file or **<Plugins>** Configuration File element. You can also define several plug-in XML files and reference each using several calls to the [addPlugin Method](#).

## Example

The following example specifies two plugin XML files for use with an instance of EditLive!.

```
var editlive = EditLiveJava("ELApplet", 700, 400);

...

editlive.addPlugin("myPlugins1.xml");
editlive.addPlugin("myPlugins2.xml");

...

editlive.show();
```

## See Also

- [Introduction to the Advanced APIs](#)
- [Simple Plugin Tutorial](#)
- [Creating Plugins Utilizing Advanced APIs Tutorial](#)



# menu (plugin)

Defining a `<menu>` element in your plugin XML will create a new menu in EditLive! called Plugins. Any `<menuItem>` or `<customMenuItem>` children elements nested in these `<menu>` elements will appear under this Plugins menu item.

If you are specifying multiple instances of plugin XML (using several calls to the [AddPlugin](#) or [AddPluginAsText](#) load-time properties or several instances of `<plugin>` nested within the Configuration File's `<Plugins>` element), any `<menuItem>` or `<customMenuItem>` elements nested in `<menu>` elements will appear under the one *Plugins* menu in EditLive!.

## Example

An instance of EditLive! uses the following load time properties:

```
...
editlive.addPlugin("myPlugin1.xml");
editlive.addPlugin("myPlugin2.xml");
...
```

File *myPlugin1.xml* contains the following elements:

```
...
<menu>
  <customMenuItem name="myPlugin1" action="raiseEvent" value="myPlugin1" text="Plugin 1" />
</menu>
...
```

File *myPlugin2.xml* contains the following elements:

```
...
<menu>
  <customMenuItem name="myPlugin2" action="raiseEvent" value="myPlugin2" text="Plugin 2" />
</menu>
...
```

This will create a Plugins menu in EditLive! that features a menu item called *Plugin 1* and a menu item called *Plugin 2*.

If you have already specified a menu called *Plugins* in your [EditLive! Configuration File](#) the Plug-in architecture will create another *Plugins* menu.

## Plug-in Element Tree Structure

`<plugin>`  
`<menu>`

```
<plugin>
  ...
  <menu>
    ...
  </menu>
</plugin>
```

## Child Elements

The `<menu>` plug-in XML element can have the exact same children as an EditLive! `<menu>` Configuration File Element.

See Also

- `<menu>` Configuration File Element

# script

Adding this element to your configuration file will have no effect in the Swing SDK.

This element is used to define Javascript files to be used in conjunction with the modified EditLive! instance. Javascript files referenced through `<script>` elements will be added to the webpage invoking the EditLive!.

## Plug-in Element Tree Structure

```
<plugin>  
<script>
```

```
<plugin>  
  ...  
  <script ... />  
</plugin>
```

## Required Attributes

src

The URL specified location of the .js file used to instantiate the modified version of EditLive!.

If the URL specified is relative, this URL will be resolved depending on which load-time property is used to instantiate the plug-in:

- [AddPlugin](#) - The relative URL will be resolved against the location of the XML plug-in file.

### Example

The following javascript occurs in a webpage loaded from <http://www.myserver.com/editlive/mywebpage.html>

```
editlive.addPlugin( "myPlugins/myPlugin.xml" );
```

myPlugin.xml contains the following `<script>` element:

```
<script src="myJavascript.js" />
```

EditLive! would attempt to cache the following file and add it to mywebpage.html: <http://www.myserver.com/editlive/myPlugins/myJavascript.js>

- [AddPluginAsText](#) - The relative URL will be resolved against the second parameter specified in this property.

### Example

The following javascript occurs in a webpage loaded from <http://www.myserver.com/editlive/editlive/mywebpage.html>

```
editlive.addPluginAsText("<?xml version=\"1.0\" ?><plugin>advancedapis jar=\"jsFiles/myJavascript.js\"  
class=\"com.ephox.footnotes.Footnotes\" /></plugin>", "http://www.myserver.com/editlive/");
```

EditLive! would attempt to cache the following file and add it to mywebpage.html: <http://www.myserver.com/editlive/jsFiles/myJavascript.js>

## Remarks

For more information on using the plugin architecture to add additional Javascript files to the webpage loading EditLive!, see the [Simple Plugin Tutorial](#).

## See Also

- [addPlugin Method](#)
- [addPluginAsText Method](#)
- [Simple Plugin Tutorial](#)

**advancedapis**

# advancedapis (Applet)

This element is used to define the java classes utilized by the EditLive! plug-in. In order to create additional java functionality for EditLive!, use the [Advanced APIs](#).

## Plug-in Element Tree Structure

```
<plugin>
<advancedapis (Applet)>
```

```
<plugin>
  <advancedapis ... />
</plugin>
```

## Required Attributes

jar

The URL specified location of the .jar file used to instantiate the modified version of EditLive!. This jar file needs to contain your [Advanced APIs](#) invocation of EditLive!.

If the URL specified is relative, this URL will be resolved depending on which load-time property is used to instantiate the plug-in:

- [AddPlugin](#) - The relative URL will be resolved against the location of the XML plug-in file.

### Example

The following Javascript occurs in a webpage loaded from <http://www.myserver.com/editlive/mywebpage.html>

```
editlive.addPlugin("myPlugins/myPlugin.xml");
```

myPlugin.xml contains the following **<advancedapis>** element:

```
<advancedapis jar="jars/myPlugin.jar" class="com.ephox.myPlugins.myPlugin" />
```

EditLive! would attempt to cache the following file: <http://www.myserver.com/editlive/myPlugins/jars/myPlugin.jar>

- [AddPluginAsText](#) - The relative URL will be resolved against the second parameter specified in this property.

### Example

The following Javascript occurs in a webpage loaded from <http://www.myserver.com/editlive/mywebpage.html>

```
editlive.addPluginAsText("<?xml version=\"1.0\" ?><plugin><advancedapis jar=\"jars/myPlugin.jar\" class=\"com.ephox.footnotes.Footnotes\" /></plugin>", "http://www.myserver.com/editlive/");
```

EditLive! for Java would attempt to cache the following file: <http://www.myserver.com/editlive/jars/myPlugin.jar>

class

The fully qualified name of the .class file used to instantiate the modified version of EditLive!. If this name is incorrect, the standard version of EditLive! will load instead of the modified version.

## Remarks

For more information on using the Advanced APIs to define a new instance of EditLive! and how to invoke this new EditLive! instance using the plugin architecture, see the [Creating Plugins Utilizing Advanced APIs Tutorial](#).

## See Also

- [addPlugin Method](#)
- [addPluginAsText Method](#)
- [Creating Plugins Utilizing Advanced APIs Tutorial](#)

# advancedapis (Swing SDK)

This element is used to define the Java classes utilized by the EditLive! for Java Swing plug-in. In order to create additional Java functionality for EditLive! for Java Swing, use the steps outlined in the [Creating and Using Plugins in the Swing SDK](#) article.

## Plug-in Element Tree Structure

```
<plugin>  
<advancedapis (Applet)>
```

```
<plugin>  
  <advancedapis ... />  
</plugin>
```

## Required Attributes

### jar

The URL specified location of the .jar file used to instantiate the modified version of EditLive! for Java Swing. This jar file needs to contain your Java code which adheres to the details specified in the [Creating and Using Plugins in the Swing SDK](#) article.

### class

The fully qualified name of the .class file used to instantiate the modified version of EditLive!. If this name is incorrect, the standard version of EditLive! will load instead of the modified version.

# Java API

Java API documentation can be found at [editlive.com/javaapi](http://editlive.com/javaapi)

Please note that the EditLive! Java API is only supported for customers who have purchased EditLive! Enterprise Edition. The following Java APIs have been removed, as of EditLive 9.1



- `com.ephox.editlive.ELJBean.setHttpLayerManager`
- `com.ephox.editlive.http.layer.HttpLayerManager`
- `com.ephox.editlive.http.layer.HttpLayer`
- `com.ephox.editlive.http.manager.HttpManager.putFile`
- `com.ephox.editlive.common.TextEvent` constants:
  - `DAV_OPEN_ACTION`
  - `DAV_SAVE_ACTION`
  - `DAV_SAVEAS_ACTION`
  - `CLEAR_STYLES_COMBO`
  - `CustomAction.SET_HTTP_LAYER`



# Menu and Toolbar Items

See: [Menu and Toolbar Item List](#)

# Tutorials

This SDK comes packaged with several tutorials, designed to teach developers how to instantiate and configure EditLive! as well as enable several of its key functionalities.

Each tutorial comes packaged with step-by-step documentation and a copy of the code for the completed tutorial exercise.

## Requirements

Several of the tutorials requires this SDK to be deployed on a specific type of web server (e.g. Java Server, IIS Server). For more information on deploying EditLive! on specified web servers, see the [EditLive! Install Guide](#).

## Contents

- [Instantiating the Editor](#)
  - [Instantiating an EditLive! Applet](#)
    - [Instantiation Tutorial](#)
    - [Instantiation Tutorial Code](#)
  - [Instantiating EditLive! in a Swing Application](#)
    - [Instantiation of a Swing Application Tutorial](#)
    - [Instantiation of a Swing Application Code](#)
- [Creating and Editing Configuration Files](#)
  - [Creating and Editing Configuration Files Tutorial](#)
  - [Specifying the Configuration File in an EditLive! Applet Tutorial](#)
  - [Specifying the Configuration File in EditLive! for Java Swing Tutorial](#)
  - [Creating and Editing Configuration Files Code](#)
- [Installing a License](#)
  - [Installing a License Tutorial](#)
  - [Installing a License Code](#)
- [Setting the Document](#)
  - [Setting the Document in the Applet](#)
    - [Setting the Document in the Applet Tutorial](#)
    - [Setting the Document in the Applet Code](#)
  - [Setting the Document in the Swing SDK](#)
    - [Setting the Document in the Swing SDK Tutorial](#)
    - [Setting the Document in the Swing SDK Code](#)
- [Setting the Body](#)
  - [Setting the Body in the Applet](#)
    - [Setting the Body in the Applet Code](#)
    - [Setting the Body in the Applet Tutorial](#)
  - [Setting the Body in the Swing SDK](#)
    - [Setting the Body in the Swing SDK Tutorial](#)
    - [Setting the Body in the Swing SDK Code](#)
- [Getting the Document](#)
  - [Getting the Document in the Applet Overview](#)
    - [Getting the Document in the Applet Tutorial](#)
    - [Getting the Document in the Applet Code](#)
  - [Getting the Document in the Swing SDK](#)
    - [Getting the Document in the Swing SDK Tutorial](#)
    - [Getting the Document in the Swing SDK Code](#)
- [Getting the Body](#)
  - [Getting the Body in the Applet](#)
    - [Getting the Body in the Applet Tutorial](#)
    - [Getting the Body in the Applet Code](#)
  - [Getting the Body in the Swing SDK](#)
    - [Getting the Body in the Swing SDK Tutorial](#)
    - [Getting the Body in the Swing SDK Code](#)
- [Adding and Removing Menu or Toolbar Items](#)
  - [Adding and Removing Menu or Toolbar Items Tutorial](#)
  - [Adding and Removing Menu or Toolbar Items Code](#)
- [Specifying Character Set](#)
  - [Specifying Character Set in the Applet](#)
    - [Specifying Character Set in the Applet Tutorial](#)
    - [Specifying the Character Set in the Applet Code](#)
      - [charEncoding.html](#)
      - [charEncoding.xml](#)
  - [Specifying Character Set in the Swing SDK](#)
    - [Specifying Character Set in the Swing SDK Tutorial](#)
    - [Specifying Character Set in the Swing SDK Code](#)
      - [charEncoding.java](#)
      - [charEncoding.xml \(Java Swing\)](#)
- [Setting CSS](#)
  - [Setting CSS in the Applet](#)
    - [Setting CSS in the Applet Tutorial](#)
    - [Setting CSS in the Applet Code](#)
  - [Setting CSS in the Swing SDK](#)
    - [Setting CSS in the Swing SDK Tutorial](#)

- Setting CSS in the Swing SDK Code
- Custom Toolbar Buttons
  - Custom Toolbar Buttons in the Applet
    - Custom Toolbar Buttons in the Applet Tutorial
    - Custom Toolbar Buttons in the Applet Code
      - customItem.xml
      - customToolbarItem.html
      - mockDialog.html
  - Custom Toolbar Buttons in the Swing SDK
    - Custom Toolbar Button in the Swing SDK Tutorial
    - Custom Toolbar Button in the Swing SDK Code
      - customItem.xml (Java Swing)
      - customToolbarItem.java
- Using Inline Editing (Tutorial)
  - Using Inline Editing Tutorial
  - Using Inline Editing Code
    - inlineEditing.html
    - posthandler.asp
- Creating Plugins Utilizing Advanced APIs
  - Creating Plugins Utilizing Advanced APIs Tutorial
  - Creating Plugins Utilizing Advanced APIs Code
    - advancedAPIPlugin.html
    - AdvancedAPIPlugin.java
    - advancedAPIPlugin.xml
- Capturing Content Before Submit
  - Capturing Content Before Submit Tutorial
  - Capturing Content Before Submit Code
- Simple Plugin
  - Simple Plugin Tutorial
  - Simple Plugin Code
    - plugin.js
    - simplePlugin.html
    - simplePlugin.xml
- Optimizing Load Times
  - Optimizing Load Times Tutorial
  - Optimizing Load Times Code
    - optimizingLoadTime.html

# Instantiating the Editor

- [Instantiating an EditLive! Applet](#)
  - [Instantiation Tutorial](#)
  - [Instantiation Tutorial Code](#)
- [Instantiating EditLive! in a Swing Application](#)
  - [Instantiation of a Swing Application Tutorial](#)
  - [Instantiation of a Swing Application Code](#)

# Instantiating an EditLive! Applet

## Overview

This tutorial provides developers with the basic knowledge required to instantiate EditLive! for Java in a web page.

## Tutorial

The [Instantiation Tutorial](#) provides a step-by-step walk-through on how to instantiate EditLive! for Java in a web page.

## Code

The complete code view for all the associated files in the [Instantiation Tutorial](#) is available [here](#).

## View the Completed Tutorial

[Click here](#) to view the completed tutorial.

# Instantiation Tutorial

## Overview

This tutorial provides developers with the basic knowledge required to instantiate EditLive! in a web page.

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

## Tutorial

### Step 1. Create a HTML page

Create a basic webpage with the following content:

```
<html>
  <body>
  </body>
</html>
```

Save this webpage as *instantiation.html*

### Step 2. Reference the editlivejava.js

EditLive! comes packaged with a directory called *redistributables*. This directory contains the core files required to instantiate EditLive! in a webpage.

EditLive! provides developers with Javascript methods to instantiate the editor to enable/disable a variety of functions. These Javascript functions are available through the *editlivejava.js* Javascript library, located in the *redistributables* directory.

Specify the *editlivejava.js* file in your *instantiation.html* webpage.

For example, if your *instantiation.html* file was in the same directory as the *redistributables* directory packaged with EditLive!, you would specify the following line of code:

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
  </body>
</html>
```

### Step 3. Define the editor name and size.

Instances of EditLive! are defined using the [EditLiveJava](#) javascript object. To define one of these objects, you need to use the [EditLiveJava](#) constructor. This constructor takes 3 parameters:

- *Name* - the unique name for this particular instance of EditLive!. In the context of this tutorial, this value is arbitrary.
- *Width* - a number specifying the width of the EditLive! instance, in pixels
- *Height* - a number specifying the height of the EditLive! instance, in pixels

The example below shows how to specify an instance of EditLive!, with the name *ELJApplet*, a width of 700 pixels, and a height of 400 pixels. This instance of the editor is assigned to a Javascript variable called *editlivejava*.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
    </script>
  </body>
</html>
```

## Step 4. Specifying the EditLive! Configuration File

In order to customize and configure various elements of the editor, developers need to create and specify an EditLive! [Configuration File](#). EditLive! Configuration Files are covered in detail in the both the [Reference](#) and [Developer Guide](#) sections of this SDK. The [Creating and Editing Configuration Files](#) tutorial describes in detail how to make and edit EditLive! Configuration Files. For the purposes of this tutorial, the example below uses the default configuration file (*sample\_eljconfig.xml*) packaged with EditLive! and stored in the *redistributables* directory. To specify a Configuration File, the [setConfigurationFile Method](#) is used.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
    </script>
  </body>
</html>
```

## Step 6. Displaying EditLive!

The [show Method](#) of EditLive! is used to to render the applet on the page.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.show();
    </script>
  </body>
</html>
```

## Code

Get the full code at [Instantiation Tutorial Code](#).

# Instantiation Tutorial Code

```
<!--
*****

instantiation.html --

EditLive! tutorial to use only the most basic
javascript methods to instantiate the editor in a webpage

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Instantiation Tutorial</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Instantiation Tutoria</h1>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELJApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlivejava = new EditLiveJava("ELJApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! for Java instance
      // at the this location.
      editlivejava.show();
    </script>
  </body>
</html>
```



# Instantiating EditLive! in a Swing Application

This tutorial provides developers with the basic knowledge required to instantiate EditLive! in a Java Swing application.

## Tutorial

The [Instantiation of a Swing Application Tutorial](#) provides a step-by-step walk-through on how to instantiate EditLive! in a Java Swing application.

## Code

The complete code view for all the associated files in the [Instantiation of a Swing Application Tutorial](#) is available [here](#).

# Instantiation of a Swing Application Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library

## Tutorial

### Step 1. Create a Java class

Create a Java class called *Instantiation*.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    public Instantiation() {
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) {
        new Instantiation();
    }
}
```

Save this class in a file called *Instantiation.java*.

### Step 2. Display a JFrame

Extend the Java class *JFrame*. Set the layout for the frame, as well as its size dimensions. Display the frame using *setVisible(true)*.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    public Instantiation() {
        super("Instantiation Tutorial");
        this.getContentPane().setLayout(new FlowLayout());
        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) {
        new Instantiation();
    }
}
```

Add a call to the UIManager class to render the frame using the native look and feel of your operating system.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    public Instantiation() {
        super("Instantiation Tutorial");
        this.getContentPane().setLayout(new FlowLayout());
        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new Instantiation();
    }
}
```

Finally, add code to allow the user to close the frame and remove the application from the JVM.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    public Instantiation() {
        super("Instantiation Tutorial");
        this.getContentPane().setLayout(new FlowLayout());
        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
        }
    }
}
```

```

        e.printStackTrace();
    }
    new Instantiation();
}
}
}

```

### Step 3. Create an Instance of the ELJBean

The EditLive! for Java bean is created using the ELJBean class. This class has a number of constructors available. For the purpose of this tutorial, we will use a constructor requiring the following parameters:

- String - the HTML Contents to be loaded into the editor.
- String - the CSS Styles to be applied to the editor's content
- int - the width, in pixels, of the editor
- int - the height, in pixels, of the editor
- File - the Configuration File to be used in conjunction with the editor.
- boolean - whether to initialise the bean or wait for an implicit call to the `init()` method to initialize the bean

Passing the boolean flag to the constructor is the preferred method creating the ELJBean. Using this constructor you can completely configure the editor before calling `init()` when you finally need the editor to appear.

In order to customize and configure various elements of the editor, developers need to create and specify an EditLive! for Java Swing Configuration File. EditLive! for Java Swing Configuration Files are covered in detail in the both the [Reference](#) and [Developer Guide](#) sections of this SDK. The [Creating and Editing Configuration Files](#) tutorial describes in detail how to make and edit EditLive! for Java Swing Configuration Files.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"), false);

    public Instantiation() {
        super("Instantiation Tutorial");
        this.getContentPane().setLayout(new FlowLayout());
        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new Instantiation();
    }
}

```

## Step 4. Display EditLive!

Now that an instance of ELJBean has been created, this bean needs to be added to the frame to be displayed.

The editor is initialized on the Swing Thread. Initializing on the Swing Thread ensures the editor will load completely as expected.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class Instantiation {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"));

    public Instantiation() throws Exception {
        super("Instantiation Tutorial");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new Instantiation();
    }
}
```

# Instantiation of a Swing Application Code

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.ephox.editlive.*;
import com.ephox.editlive.applets.*;
import com.ephox.editlive.util.guiutils.*;

import javax.jnlp.*;

/** Class loads a JFrame with a single panel, containing the
 * Ephox EditLive! Editor
 */
public class Instantiation extends JFrame{
    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
     *
     */
    public Instantiation() throws Exception {
        super("Tutorial - Instantiation");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
        }
    }
}
```

```
        e.printStackTrace();
    }
    new Instantiation();
}
}
```

# Creating and Editing Configuration Files

This tutorial provides developers with the tools and knowledge required to create and edit EditLive! Configuration Files.

EditLive! Configuration Files are used to configure various features found in the EditLive! editor. These Configuration Files are described in detail in the [Developer Guide](#) section of this SDK. The [Reference](#) Section of this SDK contains a complete element list for the possible configuration options available in a EditLive! Configuration File.

EditLive! Configuration Files can be created and/or edited with text editing tools (e.g. Microsoft Notepad). This tutorial shows how to use this method when editing Configuration Files.

## Tutorial

The [Creating and Editing Configuration Files Tutorial](#) provides a step-by-step walk-through on how to create and edit configuration files.

## Code

The [complete code view](#) for all the associated files in the [Creating and Editing Configuration Files Tutorial](#) is also available.

- [Creating and Editing Configuration Files Tutorial](#)
- [Specifying the Configuration File in an EditLive! Applet Tutorial](#)
- [Specifying the Configuration File in EditLive! for Java Swing Tutorial](#)
- [Creating and Editing Configuration Files Code](#)



# Creating and Editing Configuration Files Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Setting the Body in the Applet Tutorial](#)

## Manually Editing a Configuration File with a Text Editor

This tutorial describes how to edit Configuration Files of any variety by editing the source file itself.

### Step 1. Open the *sample\_eljconfig.xml* Configuration File

For your EditLive! SDK, open the *redistributables/editlivejava* directory in your SDK. Locate the *sample\_eljconfig.xml* Configuration File. This file is the default Configuration File provided by Ephox.

Open the *sample\_eljconfig.xml* Configuration File using a text editor.

### Step 2. Locate the `<sourceEditor>` Configuration Element

EditLive! Configuration Files use XML to store information. Use the [Reference](#) section of this SDK to read up on the `<sourceEditor>` Configuration File element.

By using the Configuration Element Tree Structure depicted in the [sourceEditor](#) article, you can see where in the Configuration File this element will be located.

Locate the `<sourceEditor>` element as it's structured in the *sample\_eljconfig.xml* file.

```
<!--  
  Specify settings for the Source (code) view of the editor  
-->  
  <sourceEditor showBodyOnly="false" />
```

### Step 3. Change the `showBodyOnly` Attribute

By changing the `showBodyOnly` attribute of `<sourceEditor>` to *false*, any instance of EditLive! that uses this Configuration File will only allow users to view and edit the contents of the `<BODY>` attribute for the HTML Document stored in EditLive!.

Change the value of the `showBodyOnly` attribute from *false* to *true*. Save the Configuration File as *config\_tutorial.xml*.

```
<!--  
  Specify settings for the Source (code) view of the editor  
-->  
  <sourceEditor showBodyOnly="true" />
```

Now you are ready to add the configuration file to the editor. To do this, go to either the [Specifying the Configuration File in an EditLive! Applet Tutorial](#) or [Specifying the Configuration File in EditLive! for Java Swing Tutorial](#).

# Specifying the Configuration File in an EditLive! Applet Tutorial

By this stage you've now created a new Configuration File that contains the configurations specified in the default Configuration File, as well as the following changes:

- The Design and Code view tabs are located at the top of the editor.
- When editing content in the Code view, only the HTML content in the <BODY> of the editor's HTML Document is shown.

This configuration of EditLive! works well with the [Setting the Body in the Applet Tutorial](#). Here is an example of the *config\_tutorial.xml* configuration file used in conjunction with the [Setting the Body in the Applet Code](#):

This code infers that the *config\_tutorial.xml* file is stored in the same location as the default Configuration File *sample\_eljconfig.xml*.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Set <BODY> Contents" onclick="buttonPress()"><
/p>
      <script src="../../redistributables/editlivejava/editlivejava.js"
        language="JavaScript"></script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/config_tutorial.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.show();

        function buttonPress() {
          editlivejava.setBody(encodeURIComponent(document.exampleForm.
bodyContents.value));
        }
      </script>
    </form>
  </body>
</html>
```

See also:

- [Creating and Editing Configuration Files Tutorial](#)

# Specifying the Configuration File in EditLive! for Java Swing Tutorial

By this stage you've now created a new Configuration File that contains the configurations specified in the default Configuration File, as well as the following changes:

- The Design and Code view tabs are located at the top of the editor.
- When editing content in the Code view, only the HTML content in the <BODY> of the editor's HTML Document is shown.

This configuration of EditLive! for Java Swing works well with the [Setting the Body Tutorial](#). Here is an example of the *config\_tutorial.xml* configuration file used in conjunction with the [Setting the Body Code](#):

This code infers that the *config\_tutorial.xml* file is stored in the same location as the default Configuration File *sample\_eljconfig.xml*.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.ephox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to populate the <BODY> of
 * a Document stored in EditLive!, at both load-time and run-time.
 *
 */
public class Config extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 300, getClass().getResource(
    "config_tutorial.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public Config() throws Exception {
        super("Tutorial - Creating and Editing Configuration Files");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);
        // add listener to button
        bodyButton.addActionListener(this);
    }
}
```

```

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // specify textarea content
        source.setText("<p>Text Area Contents.<br/><b>Note: </b>Ensure this text area contains correctly
formatted <i>HTML</i></p>");

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 640));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
     *
     * @param e ActionEvent sent by JButton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == bodyButton) {
            try {
                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        editLiveBean.setBody(source.getText());
                    }
                });
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }

        new Config();
    }
}

```

# Creating and Editing Configuration Files Code

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

<!--

This file customizes and configures EditLive!.
TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--
        <base href="http://www.youserver.com/cms/" />
        -->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in
HTML
        -->
        <!--
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        -->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!--
        <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/>
        -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFF0CC;
          }
        </style>
      </html>
    </document>
  </editlive>

```

```

-->
</head>

<!--
    Default document body. Add content here if you want this to be the default when the editor
    loads, although this is better done at runtime.
-->
<body>
</body>

</html>
</document>

<!--
    Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-4DC5-EDF4-2486"
    licensee="For Evaluation Only"
    release="8.0"
    type="Evaluation License"
    productivityPack="true"
  />
</ephoxLicenses>

<!--
    Specify the location of the spell checker and thesaurus.
    If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
    based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
    Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
    Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the
    tabs.
-->
<wysiwygEditor
  tabPlacement="top"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false">
  enableTrackChanges="false"
<!--
    Define Custom Tags actions
-->
<!--
<customTags>
  <doubleClickActions>
    <action.../>
  </doubleClickActions>

```

```

</customTags>
-->

<!--
    Define additional symbols for the symbol dialog here
-->
<!--
<symbols>
</symbols>
-->
</wysiwygEditor>

<!--
    Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="true"/>

<!--
    Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
    Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
    <!--
        Specify HTTP upload settings
        'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
        'href' is your server-side script for handling multipart-formdata uploads from ELJ
    -->
    <httpUpload
        base="http://www.yourserver.com/userfiles/"
        href="http://www.yourserver.com/scripts/upload.jsp">

        <!--
            Specify any additional fields to post with the image data
        -->
        <!--<
            httpUploadData name="hello" data="world"/>
        -->

    </httpUpload>

    <images allowLocalImages="true" allowUserSpecified="true">
        <!--
            The list of images which appear in the Insert Image dialog.
            TIP: Dynamically generate this from your database or repository to achieve an easy image
library.
        -->
        <imageList>
            <image name="EditLive!"
                description="EditLive! Logo"
                alt="EditLive! Logo"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
                title="EditLive!" />

            <image name="iMac"
                alt="iMac Computer"
                description="iMac Computer"
                title="iMac"
                border="0"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

            <image name="Apple Computer"
                alt="Apple Computer"
                title="Apple Computer"
                description="Picture of a new Apple Computer"

```

```

border="0"
src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

<image name="IBM Thinkpad"
alt="IBM Thinkpad"
border="0"
title="IBM Thinkpad"
description="Picture of a new IBM Thinkpad"
src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"
/>

</imageList>
</images>
<multimedia>
  <types>
    <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
      <param name="movie" />
      <param name="quality" />
      <param name="bgcolor" />
    </type>
    <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
      <param name="autohref" />
      <param name="autoplay" />
      <param name="bgcolor" />
      <param name="cache" />
      <param name="controller" />
      <param name="correction" />
      <param name="dontflattenwhensaving" />
      <param name="enablejavascript" />
      <param name="endtime" />
      <param name="fov" />
      <param name="height" />
      <param name="href" />
      <param name="kioskmode" />
      <param name="loop" />
      <param name="movieid" />
      <param name="moviename" />
      <param name="node" />
      <param name="pan" />
      <param name="playeveryframe" />
      <param name="qtsrcchokespeed" />
      <param name="scale" />
      <param name="starttime" />
      <param name="target" />
      <param name="targetcache" />
      <param name="tilt" />
      <param name="urlsubstitute" />
      <param name="volume" />
    </type>
    <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
      <param name="animationAtStart" />
      <param name="autoStart" />
      <param name="showControls" />
      <param name="clickToPlay" />
      <param name="transparentAtStart" />
    </type>
    <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
      <param name="animationAtStart" />
      <param name="autoStart" />
      <param name="showControls" />
      <param name="clickToPlay" />
      <param name="transparentAtStart" />
    </type>
    <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
    <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
    <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
  </types>
</multimedia>

```



```

</mediaSettings>

<hyperlinks>

  <hyperlinkList>
    <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
    <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
    <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
  </hyperlinkList>

  <mailtoList>
    <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
  </mailtoList>

</hyperlinks>

<!--
  Customize the EditLive! menus

  Note: you must display some sort of Ephox copyright statement within your application, only
  remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
  copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

  <menu name="ephox_filemenu">
    <menuItem name="New"/>
    <menuItem name="Open"/>
    <menuSeparator/>
    <menuItem name="Save"/>
    <menuItem name="SaveAs"/>
    <menuSeparator/>
    <menuItem name="Print"/>
  </menu>

  <menu name="ephox_editmenu">
    <menuItem name="Undo"/>
    <menuItem name="Redo"/>
    <menuSeparator/>
    <menuItem name="Cut"/>
    <menuItem name="Copy"/>
    <menuItem name="Paste"/>
    <menuItem name="PasteSpecial"/>
    <menuSeparator/>
    <menuItem name="Select"/>
    <menuItem name="SelectAll"/>
    <menuSeparator/>
    <menuItem name="Find"/>
    <menuSeparator/>
  </menu>

  <menu name="ephox_viewmenu">
    <menuItemGroup name="SourceView"/>
    <menuSeparator/>
    <menuItem name="Popout"/>
    <menuSeparator/>
    <menuItem name="showDocumentNavigator"/>
    <menuSeparator/>
    <menuItem name="ParagraphMarker"/>
  </menu>

  <menu name="ephox_insertmenu">

    <menuItem name="HLink"/>
    <menuItem name="Bookmark"/>
    <menuItem name="RemoveHyperlink" />
    <menuSeparator/>
    <menuItem name="ImageServer"/>
    <menuItem name="InsertObject" />

```

```

    <menuSeparator/>
    <menuItem name="Symbol"/>
    <menuItem name="HRule"/>
    <menuSeparator/>
    <menuItem name="DateTime"/>
    <menuSeparator/>
    <menuItem name="insertcomment"/>
</menu>

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>

  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>

  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>
  <menuItem name="InsTextField"/>
  <menuItem name="InsPasswordField"/>
  <menuItem name="InsHiddenField"/>
  <menuItem name="InsFileField"/>
  <menuItem name="InsButtonField"/>
  <menuItem name="InsSubmitField"/>

```

```

        <menuItem name="InsResetField"/>
        <menuItem name="InsCheckboxField"/>
        <menuItem name="InsRadioField"/>
        <menuItem name="InsTextAreaField"/>
        <menuItem name="InsSelectField"/>
        <menuItem name="InsImageField"/>
    </menu>
    <menu name="ephox_trackchangesmenu">
        <menuItem name="enabletrackchanges" />
        <menuSeparator />
        <menuItem name="acceptChange" />
        <menuItem name="rejectChange" />
        <menuSeparator />
        <menuItem name="previousChange" />
        <menuItem name="nextChange" />
        <menuSeparator />
        <menuItem name="acceptAllChanges" />
        <menuItem name="rejectAllChanges" />
        <menuSeparator />
        <menuItem name="showTrackChangesDialog" />
        <menuSeparator />
        <menuItem name="setUsername" />
    </menu>
</menuBar>

<!--
    Customize the EditLive! toolbars
-->
<toolbars>
    <toolbar name="Command">

        <toolbarButton name="Print"/>
        <toolbarSeparator/>
        <toolbarButton name="Spelling"/>

        <toolbarButton name="Find"/>
        <toolbarSeparator/>
        <toolbarButton name="Cut"/>
        <toolbarButton name="Copy"/>
        <toolbarButton name="Paste"/>
        <toolbarButton name="FormatPainter" />
        <toolbarSeparator/>
        <toolbarButton name="Undo"/>
        <toolbarButton name="Redo"/>
        <toolbarSeparator/>
        <toolbarButton name="HLink"/>
        <toolbarButton name="ImageServer"/>
        <toolbarButton name="insertequation"/>
        <toolbarSeparator/>
        <toolbarButton name="InsTableWizard"/>
        <toolbarButton name="InsRow"/>
        <toolbarButton name="InsCol"/>
        <toolbarButton name="DelRow"/>
        <toolbarButton name="DelCol"/>
        <toolbarSeparator/>
        <toolbarButton name="enableTrackChanges"/>
        <toolbarButton name="acceptChange"/>
        <toolbarButton name="rejectchange"/>
        <toolbarButton name="previouschange"/>
        <toolbarButton name="nextchange"/>
        <toolbarSeparator/>
        <toolbarButton name="ParagraphMarker"/>
        <toolbarSeparator/>

        <toolbarButton name="Popout"/>
    </toolbar>

```

```

<toolbar name="Format">
  <!--
    Styles from any embedded or external stylesheets will also be automatically added to the
    Styles drop-down
  -->
  <toolbarComboBox name="Style">
    <comboBoxItem name="P" />
    <comboBoxItem name="H1" />
    <comboBoxItem name="H2" />
    <comboBoxItem name="H3" />
    <comboBoxItem name="H4" />
    <comboBoxItem name="H5" />
    <comboBoxItem name="H6" />
  </toolbarComboBox>

  <!--
    You can remove the Font drop-down if you just want users to use Styles.
    The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac
    OS X and Windows
  -->
  <toolbarComboBox name="Face">
    <comboBoxItem name="Arial" text="Arial" />
    <comboBoxItem name="Arial Black" text="Arial Black" />
    <comboBoxItem name="Arial Narrow" text="Arial Narrow" />
    <comboBoxItem name="Comic Sans MS" text="Comic Sans MS" />
    <comboBoxItem name="Courier New" text="Courier New" />
    <comboBoxItem name="Georgia" text="Georgia" />
    <comboBoxItem name="Impact" text="Impact" />
    <comboBoxItem name="Times New Roman" text="Times New Roman" />
    <comboBoxItem name="Trebuchet MS" text="Trebuchet MS" />
    <comboBoxItem name="Verdana" text="Verdana" />
  </toolbarComboBox>

  <!--
    Font Size drop-down
  -->
  <toolbarComboBox name="Size">
    <comboBoxItem name="1" text="8pt" />
    <comboBoxItem name="2" text="10pt" />
    <comboBoxItem name="3" text="12pt" />
    <comboBoxItem name="4" text="14pt" />
    <comboBoxItem name="5" text="18pt" />
    <comboBoxItem name="6" text="24pt" />
    <comboBoxItem name="7" text="36pt" />
  </toolbarComboBox>
  <toolbarSeparator />
  <toolbarButton name="Bold" />
  <toolbarButton name="Italic" />
  <toolbarButton name="Underline" />
  <toolbarSeparator />
  <toolbarButtonGroup name="Align" />
  <toolbarSeparator />
  <toolbarButtonGroup name="List" />
  <toolbarButton name="DecreaseIndent" />
  <toolbarButton name="IncreaseIndent" />
  <toolbarSeparator />
  <toolbarButton name="HighlightColor" />
  <toolbarButton name="Color" />
</toolbar>
</toolbars>

<!--
  Customize the EditLive! shortcut menu
-->
<shortcutMenu>
  <shrtMenuItem name="Undo" />
  <shrtMenuItem name="Redo" />
  <shrtMenuSeparator />
  <shrtMenuItem name="Cut" />

```

```
<shrtMenuItem name="Copy" />
<shrtMenuItem name="Paste" />
<shrtMenuSeparator />
<shrtMenuItem name="Select" />
<shrtMenuSeparator />
<shrtMenuItem name="acceptChange" />
<shrtMenuItem name="rejectChange" />
<shrtMenuItem name="nextchange" />
<shrtMenuItem name="previouschange" />
<shrtMenuSeparator />
<shrtMenuItem name="Hyperlink" />
<shrtMenuItem name="RemoveHyperlink" />
<shrtMenuItem name="PropImage" />
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator />
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator />
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator />
  <shrtMenuItem name="synonyms" />
  <shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# Installing a License

The purpose of this tutorial is to teach developers how to install their EditLive! license.

When purchasing an EditLive! license from Tiny you will be sent an email containing your license in an *editlive.lic* file.

To activate an EditLive! license, the information in your *editlive.lic* file needs to be stored in an EditLive! Configuration File to be used by the editor. EditLive! Configuration Files can be edited using a text editing application (such as Microsoft Notepad). This tutorial describes how to use this method for installing an EditLive! license.

## Tutorial

The [Installing a License Tutorial](#) provides a step-by-step walk-through on how to create and edit configuration files.

For the purpose of this tutorial, the assumption is that you wish to apply your license to the default Configuration File packaged with EditLive!. The steps outlined in this tutorial can be applied to any configuration file.

## Code

The complete code view for all the associated files in the [Installing a License Tutorial](#) is available [here](#).

- [Installing a License Tutorial](#)
- [Installing a License Code](#)

# Installing a License Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Creating and Editing Configuration Files Tutorial](#)

## Installing a License Using a Text Editor

### Step 1. Open the `sample_eljconfig.xml` Configuration File

For your EditLive! SDK, open the `redistributables/edlitlivejava` directory in your SDK. Locate the `sample_eljconfig.xml` Configuration File. This file is the default Configuration File provided by Ephox.

Open the `sample_eljconfig.xml` Configuration File using a text editor.

### Step 2. Locate the `<ephoxLicenses>` Configuration Element

EditLive! for Java Swing Configuration Files use XML to store information. Use the [Reference](#) section of this SDK to read up on the `<ephoxLicenses>` Configuration File element.

By using the Configuration Element Tree Structure depicted in the `<ephoxLicenses>` article, you can see where in the Configuration File this element will be located.

Locate the `<ephoxLicenses>` element as it is structured in the `sample_eljconfig.xml` file.

```
<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-4DC5-EDF4-2486"
    licensee="For Evaluation Only"
    release="8.0"
    type="Evaluation License"
    productivityPack="true"
  />
</ephoxLicenses>
```

### Step 3. Add the `<license>` Element from your License File

Your license file will contain a `<license>` element. To add your license to the configuration file, copy the `<license>` element from the license file into the `<ephoxLicenses>` element, under the current `<license>` element.

For example, if your license file contained the following:

```
<ephoxLicenses>
  <license
    domain="MY.SERVER.COM"
    key="5FFF-635E-395E-E5F3"
    licensee="ME"
    type="Unlimited Subdomain"
    release="8.0"
    productivityPack="true"
  />
</ephoxLicenses>
```

The `<ephoxLicenses>` element in your configuration file should appear as follows:

```
<!--  
Add your Ephox-provided license key here  
-->  
<ephoxLicenses>  
  <license  
    domain="LOCALHOST"  
    key="6FFF-4DC5-EDF4-2486"  
    licensee="For Evaluation Only"  
    release="8.0"  
    type="Evaluation License"  
    productivityPack="true"  
  />  
  <license  
    domain="MY.SERVER.COM"  
    key="5FFF-635E-395E-E5F3"  
    licensee="ME"  
    type="Unlimited Subdomain"  
    release="8.0"  
    productivityPack="true"  
  />  
</ephoxLicenses>
```



# Installing a License Code

If your license file contained the following:

```
<ephoxLicenses>
  <license
    domain="MY.SERVER.COM"
    key="5FFF-635E-395E-E5F3"
    licensee="ME"
    type="Unlimited Subdomain"
    release="8.0"
    productivityPack="true"
  />
</ephoxLicenses>
```

If you are using the *sample\_eljconfig.xml* default configuration file with your instance of EditLive!, it would appear as follows after applying the license:

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

<!--

This file customizes and configures EditLive!.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings.

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
```

```

        border-bottom: solid 1px #003366;
    }
    p.fineprint{
        font-size: 8pt;
        text-align: center;
    }
    span.comment {
        border: solid 1px #FFFF00;
        background-color: #FFFFCC;
    }
</style>
-->
</head>

<!--
Default document body. Add content here if you want this to be the default when the editor
loads, although this is better done at runtime.
-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-4DC5-EDF4-2486"
    licensee="For Evaluation Only"
    release="8.0"
    type="Evaluation License"
    productivityPack="true"
  />
  <license
    domain="MY.SERVER.COM"
    key="5FFF-635E-395E-E5F3"
    licensee="ME"
    type="Unlimited Subdomain"
    release="8.0"
    productivityPack="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

```

```

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
    tabPlacement="bottom"
    brOnEnter="false"
    showDocumentNavigator="false"
    disableInlineImageResizing="false"
    disableInlineTableResizing="false"
    enableTrackChanges="false"
>
    <!-- Define Custom Tags actions -->
    <!--
    <customTags>
        <doubleClickActions>
            <action.../>
        </doubleClickActions>
    </customTags>
    -->
    <!-- Define additional symbols for the symbol dialog here -->
    <!--
    <symbols></symbols>
    -->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
    <!--
    Specify HTTP upload settings
    'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
    'href' is your server-side script for handling multipart-formdata uploads from ELJ
    The httpUploadData element specifies any additional fields to post with the image data
    -->
    <!--
    <httpUpload
        base="http://www.yourserver.com/userfiles/"
        href="http://www.yourserver.com/scripts/upload.jsp">
        <httpUploadData name="hello" data="world"/>
    </httpUpload>
    -->

    <images allowLocalImages="true" allowUserSpecified="true">
        <!--
        The list of images which appear in the Insert Image dialog.
        TIP: Dynamically generate this from your database or repository to achieve an easy image library.
        -->

        <imageList>
            <image name="EditLive!"
                description="EditLive! Logo"
                alt="EditLive! Logo"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
                title="EditLive!" />

            <image name="iMac"
                alt="iMac Computer"

```

```

description="iMac Computer"
title="iMac"
border="0"
src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

<image name="Apple Computer"
alt="Apple Computer"
title="Apple Computer"
description="Picture of a new Apple Computer"
border="0"
src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

<image name="IBM Thinkpad"
alt="IBM Thinkpad"
border="0"
title="IBM Thinkpad"
description="Picture of a new IBM Thinkpad"
src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"
/>

</imageList>
</images>
<multimedia>
<types>
<type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
<param name="movie" />
<param name="quality" />
<param name="bgcolor" />
</type>
<type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
<param name="autohref" />
<param name="autoplay" />
<param name="bgcolor" />
<param name="cache" />
<param name="controller" />
<param name="correction" />
<param name="dontflattenwhensaving" />
<param name="enablejavascript" />
<param name="endtime" />
<param name="fov" />
<param name="height" />
<param name="href" />
<param name="kioskmode" />
<param name="loop" />
<param name="movieid" />
<param name="moviename" />
<param name="node" />
<param name="pan" />
<param name="playeveryframe" />
<param name="qtsrccchokespeed" />
<param name="scale" />
<param name="starttime" />
<param name="target" />
<param name="targetcache" />
<param name="tilt" />
<param name="urlsubstitute" />
<param name="volume" />
</type>
<type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
<param name="animationAtStart" />
<param name="autoStart" />
<param name="showControls" />
<param name="clickToPlay" />
<param name="transparentAtStart" />
</type>
<type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
<param name="animationAtStart" />
<param name="autoStart" />

```

```

        <param name="showControls" />
        <param name="clickToPlay" />
        <param name="transparentAtStart" />
    </type>
    <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
    <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
    <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
</types>
</multimedia>
</mediaSettings>

<hyperlinks>

    <hyperlinkList>
        <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
        <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
        <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
    </hyperlinkList>

    <mailtoList>
        <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
    </mailtoList>

</hyperlinks>

<!--
Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

    <menu name="ephox_filemenu">
        <menuItem name="New" />
        <menuItem name="Open" />
        <menuSeparator />
        <menuItem name="Save" />
        <menuItem name="SaveAs" />
        <menuSeparator />
        <menuItem name="Print" />
    </menu>

    <menu name="ephox_editmenu">
        <menuItem name="Undo" />
        <menuItem name="Redo" />
        <menuSeparator />
        <menuItem name="Cut" />
        <menuItem name="Copy" />
        <menuItem name="Paste" />
        <menuItem name="PasteSpecial" />
        <menuSeparator />
        <menuItem name="Select" />
        <menuItem name="SelectAll" />
        <menuSeparator />
        <menuItem name="Find" />
        <menuSeparator />
    </menu>

    <menu name="ephox_viewmenu">
        <menuItemGroup name="SourceView" />
        <menuSeparator />
        <menuItem name="Popout" />
        <menuSeparator />
        <menuItem name="showDocumentNavigator" />
        <menuSeparator />
        <menuItem name="ParagraphMarker" />
    </menu>

```

```

<menu name="ephox_insertmenu">
  <menuItem name="HLink"/>
  <menuItem name="Bookmark"/>
  <menuItem name="RemoveHyperlink" />
  <menuSeparator/>
  <menuItem name="ImageServer"/>
  <menuItem name="InsertObject"/>
  <menuSeparator/>
  <menuItem name="Symbol"/>
  <menuItem name="HRule"/>
  <menuSeparator/>
  <menuItem name="DateTime"/>
  <menuSeparator/>
  <menuItem name="insertcomment"/>
</menu>

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>
  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>
  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuItem name="tableautofit"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>

```

```

        <menuItem name="InsTextField"/>
        <menuItem name="InsPasswordField"/>
        <menuItem name="InsHiddenField"/>
        <menuItem name="InsFileField"/>
        <menuItem name="InsButtonField"/>
        <menuItem name="InsSubmitField"/>
        <menuItem name="InsResetField"/>
        <menuItem name="InsCheckboxField"/>
        <menuItem name="InsRadioField"/>
        <menuItem name="InsTextAreaField"/>
        <menuItem name="InsSelectField"/>
        <menuItem name="InsImageField"/>
    </menu>
    <menu name="ephox_trackchangesmenu">
        <menuItem name="enabletrackchanges" />
        <menuSeparator />
        <menuItem name="acceptChange" />
        <menuItem name="rejectChange" />
        <menuSeparator />
        <menuItem name="previousChange" />
        <menuItem name="nextChange" />
        <menuSeparator />
        <menuItem name="acceptAllChanges" />
        <menuItem name="rejectAllChanges" />
        <menuSeparator />
        <menuItem name="showTrackChangesDialog" />
        <menuSeparator />
        <menuItem name="setUsername" />
    </menu>
</menuBar>

<!--
Customize the EditLive! toolbars
-->
<toolbars>
    <toolbar name="Command">
        <toolbarButton name="Print"/>
        <toolbarSeparator/>
        <toolbarButton name="Spelling"/>
        <toolbarButton name="Find"/>
        <toolbarSeparator/>
        <toolbarButton name="Cut"/>
        <toolbarButton name="Copy"/>
        <toolbarButton name="Paste"/>
        <toolbarButton name="FormatPainter" />
        <toolbarSeparator/>
        <toolbarButton name="Undo"/>
        <toolbarButton name="Redo"/>
        <toolbarSeparator/>
        <toolbarButton name="HLink"/>
        <toolbarButton name="ImageServer"/>
        <toolbarButton name="insertequation"/>
        <toolbarSeparator/>

        <toolbarButton name="InsTableWizard"/>
        <toolbarButton name="InsRow"/>
        <toolbarButton name="InsCol"/>
        <toolbarButton name="DelRow"/>
        <toolbarButton name="DelCol"/>
        <toolbarSeparator/>
        <toolbarButton name="enableTrackChanges"/>
        <toolbarButton name="acceptChange"/>
        <toolbarButton name="rejectchange"/>
        <toolbarButton name="previouschange"/>
        <toolbarButton name="nextchange"/>
        <toolbarSeparator/>
        <toolbarButton name="ParagraphMarker"/>
        <toolbarSeparator/>
        <toolbarButton name="Popout"/>
    </toolbar>

```

```

<toolbar name="Format">
  <!--
drop-down
  Styles from any embedded or external stylesheets will also be automatically added to the Styles
  -->
  <!--
  -->
  <toolbarComboBox name="Style">
    <comboBoxItem name="P"/>
    <comboBoxItem name="H1"/>
    <comboBoxItem name="H2"/>
    <comboBoxItem name="H3"/>
    <comboBoxItem name="H4"/>
    <comboBoxItem name="H5"/>
    <comboBoxItem name="H6"/>
  </toolbarComboBox>
  <!--
  You can remove the Font drop-down if you just want users to use Styles.
  The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
and Windows
  To change the default font, change the embedded style sheet in the 'style' element above.
  -->
  <toolbarComboBox name="Face">
    <comboBoxItem name="Arial" text="Arial"/>
    <comboBoxItem name="Arial Black" text="Arial Black"/>
    <comboBoxItem name="Arial Narrow" text="Arial Narrow"/>
    <comboBoxItem name="Comic Sans MS" text="Comic Sans MS"/>
    <comboBoxItem name="Courier New" text="Courier New"/>
    <comboBoxItem name="Georgia" text="Georgia"/>
    <comboBoxItem name="Impact" text="Impact"/>
    <comboBoxItem name="Times New Roman" text="Times New Roman"/>
    <comboBoxItem name="Trebuchet MS" text="Trebuchet MS"/>
    <comboBoxItem name="Verdana" text="Verdana"/>
  </toolbarComboBox>
  <!--
  Font Size drop-down
  -->
  <toolbarComboBox name="Size">
    <comboBoxItem name="1" text="8pt"/>
    <comboBoxItem name="2" text="10pt"/>
    <comboBoxItem name="3" text="12pt"/>
    <comboBoxItem name="4" text="14pt"/>
    <comboBoxItem name="5" text="18pt"/>
    <comboBoxItem name="6" text="24pt"/>
    <comboBoxItem name="7" text="36pt"/>
  </toolbarComboBox>
  <toolbarSeparator/>
  <toolbarButton name="Bold"/>
  <toolbarButton name="Italic"/>
  <toolbarButton name="Underline"/>
  <toolbarSeparator/>
  <toolbarButtonGroup name="Align"/>
  <toolbarSeparator/>
  <toolbarButtonGroup name="List"/>
  <toolbarButton name="DecreaseIndent"/>
  <toolbarButton name="IncreaseIndent"/>
  <toolbarSeparator/>
  <toolbarButton name="HighlightColor"/>
  <toolbarButton name="Color"/>
</toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
  <shrtMenu>
    <shrtMenuItem name="Undo"/>
    <shrtMenuItem name="Redo"/>
    <shrtMenuSeparator/>
    <shrtMenuItem name="Cut"/>
    <shrtMenuItem name="Copy"/>
    <shrtMenuItem name="Paste"/>
  </shrtMenu>
</shortcutMenu>

```



```
<shrtMenuSeparator/>
<shrtMenuItem name="Select" />
<shrtMenuSeparator/>
<shrtMenuItem name="acceptChange" />
<shrtMenuItem name="rejectChange" />
<shrtMenuItem name="nextchange" />
<shrtMenuItem name="previouschange" />
<shrtMenuSeparator/>
<shrtMenuItem name="Hyperlink" />
<shrtMenuItem name="RemoveHyperlink" />
<shrtMenuItem name="PropImage" />
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator/>
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator/>
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator/>
    <shrtMenuItem name="synonyms" />
<shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# Setting the Document

- [Setting the Document in the Applet](#)
  - [Setting the Document in the Applet Tutorial](#)
  - [Setting the Document in the Applet Code](#)
- [Setting the Document in the Swing SDK](#)
  - [Setting the Document in the Swing SDK Tutorial](#)
  - [Setting the Document in the Swing SDK Code](#)

# Setting the Document in the Applet

This tutorial provides developers with the knowledge required to set the HTML Document stored in EditLive!. This tutorial depicts how the HTML Document of EditLive! can be set at both load-time or run-time.

## Tutorial

The [Setting the Document in the Applet Tutorial](#) provides a step-by-step walk-through on how to specify the HTML Document stored in an EditLive! applet.

## Code

The complete code view for all the associated files in the [Setting the Document in the Applet Tutorial](#) is available [here](#).

# Setting the Document in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! in a Webpage

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! in a webpage.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.show();
    </script>
  </body>
</html>
```

Save this webpage as *setDocument.html*

### Step 2. Setting the HTML Document to Appear When the Editor Loads

The [setDocument Method](#) is used to set the contents of the HTML Document stored in EditLive!.

For the purpose of this tutorial, we will load the following HTML into the editor:

```
<p>Original <i>HTML</i> loaded into EditLive!</p>
```

When specifying any HTML to insert into EditLive!, this HTML must be URL Encoded. Javascript provides numerous URL Encoding methods (such as *escape* and *encodeURIComponent*). If using a server-side language to create the webpage where EditLive! is instantiated, Tiny advises using the URL encoding method of this language to encode the HTML to be used with either the load-time properties or run-time functions of EditLive!.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setDocument(encodeURIComponent("<html><head><title>Default Document Title<
/title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 3. Create a HTML Textarea

Add a HTML textarea to the webpage. The purpose of this textarea will be to allow users to write HTML content, which they can then send to EditLive! at any time.

```
<html>
  <body>
    <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 4. Add a Button to Copy the Textarea Contents into EditLive!

In order to copy the contents of the textarea into EditLive!, a button should be created to call a Javascript method. This Javascript method will be written in the next step to perform the copying action.

```
<html>
  <body>
    <p><textarea id="documentContents" cols="80" rows="5"></textarea>
      <br/><input type="button" value="Set Document Contents" onclick="buttonPress()"></p>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 5. Create a Javascript Function to Set the HTML Document of EditLive! with the Textarea Contents

The [setDocument Method \(Run Time\)](#) can be used to set the HTML Document of EditLive!. You can directly reference the textarea's content by using the Javascript DOM methods.

As seen in Step 2, a URL encoding function is required when passing HTML to EditLive!.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Set Document Contents" onclick="buttonPress()"></p>
      <script src="../../redistributables/editlivejava/editlivejava.js"
        language="JavaScript"></script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
      </script>
    </form>
  </body>
</html>
```

```
        editlivejava.show();

        function buttonPress() {
            editlivejava.setDocument(encodeURIComponent(document.exampleForm.
documentContents.value));
        }
    </script>
</form>
</body>
</html>
```

#### See Also

- [<sourceEditor>](#) Configuration Element
- [setDocument Method](#)
- [getDocument Method](#)

# Setting the Document in the Applet Code

```
<!--
*****

setDocument.html --

This tutorial shows developers how to populate the HTML
Document stored in EditLive!, at both load-time and run-time.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Setting the Editor's HTML Document - Tutorial</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Setting the Editor's HTML Document</h1>

    <form name="exampleForm">

      <p>This tutorial shows how to populate the HTML Document stored in EditLive!</p>

      <p>This tutorial also shows how to extract the contents of a webpage field and store
it's contents in EditLive! HTML Document </p>

      <!--
      The textarea used to load HTML content into EditLive!'s HTML document.
      -->
      <textarea id="documentContents" cols="80" rows="5"><html><head><title>New Document Title<
/title></head><body><p>A New HTML Document to load into <b>EditLive!</b></p></body></html></textarea>

      <!--
      The button for copying the content from the textarea to EditLive!
      -->
      <p>Pressing this button will copy the HTML contents of the above textarea into the &lt;
BODY&gt; of the Document stored in EditLive!<br/>
      <input type="button" value="Set HTML Contents" onclick="buttonPress()"></p>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");

        // Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.

        // Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
        editlive.setDocument(encodeURIComponent("<html><head><title>Default Document
Title</title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>"));

        // .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
```

```

// at the this location.
editlive.show();

/** Function extracts the contents of the 'bodyContents' textarea and sets this
HTML into
    * the <body> field of the EditLive! HTML Document.
    */
function buttonPress() {
URL Encoded.
    // Before sending HTML to the instance of EditLive!, this HTML must be

    // Javascript provides several URL Encoding methods, the best of which is
    // 'encodeURIComponent'
    editlive.setDocument(encodeURIComponent(document.exampleForm.
documentContents.value));
    }
    </script>

    </form>
    </body>
</html>
```



# Setting the Document in the Swing SDK

This tutorial provides developers with the knowledge required to set the HTML Document stored in EditLive!. This tutorial depicts how the HTML Document of EditLive! can be set at both load-time or run-time.

## Tutorial

The [Setting the Document in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to specify the HTML Document stored in an EditLive! applet.

## Code

The complete code view for all the associated files in the [Setting the Document in the Swing SDK Tutorial](#) is available [here](#).

- [Setting the Document in the Swing SDK Tutorial](#)
- [Setting the Document in the Swing SDK Code](#)

# Setting the Document in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! for Java Swing in a JFrame

As shown in the Instantiation Tutorial, create an instance of EditLive! for Java Swing in a JFrame.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"), false);

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

```

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetDocument();
}
}

```

## Step 2. Setting the HTML Document to Appear When the Editor Loads

The [ELJBean](#) constructor can accept a HTML string to load into the Editor. This string can either be a complete HTML Document or simply the contents of the <BODY> element.

For the purpose of this tutorial, we will load the following complete HTML into the EditLive! for Java Swing editor:

```

<html><head><title>Default Document Title</title></head><body><p>Original <i>HTML Document</i> loaded into
EditLive!</p></body></html>

```

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}

```

```

}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetDocument();
}
}

```

### Step 3. Create a JTextArea and Add it to the JFrame

Add a [JTextArea](#) to the webpage. The purpose of this JTextArea will be to allow users to write HTML content, which they can then send to EditLive! for Java Swing at any time.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument {
    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><<
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);
    }
}

```

```

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetDocument();
    }
}

```

## Step 4. Add a Button to Copy the JTextarea Contents into EditLive! for Java Swing

In order to copy the contents of the JTextArea into EditLive! for Java Swing, a [JButton](#) should be created to trigger the copying event. The listener for the button's click action will be added in the next step.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument {
    /** Buttons used to get html contents of JTextArea and set into EditLive! */
    private JButton documentButton = new JButton("Set HTML Document");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });
    }

    // Create a JPanel to hold the ELJBean

```

```

JPanel editorHolder = new JPanel(new FlowLayout());
editorHolder.add(editLiveBean);
// Add the JPanel to the JFrame
this.getContentPane().add(editorHolder);

// Create a JPanel to hold the button and the text area
JPanel buttonAndText = new JPanel(new BorderLayout());

// create a JPanel to hold the button
JPanel buttonHolder = new JPanel(new FlowLayout());
buttonHolder.add(documentButton);

// add button holding panel
buttonAndText.add(buttonHolder, BorderLayout.NORTH);

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 620));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetDocument();
}
}

```

## Step 5. Extend the ActionListener and Add an ActionListener to the JButton

By implementing the [ActionListener](#) interface this class can listen to [ActionEvents](#). Add this class as an ActionListener to the JButton.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument implements ActionListener {
    /** Buttons used to get html contents of JTextArea and set into EditLive */
    private JButton documentButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

```

```

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);
        // add listener to button
        documentButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
    *
    * @param args the command line arguments - ignored
    */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetDocument();
    }
}

```

## Step 6. Implementing the actionPerformed Method

Finally, to catch events fired by the JButton, implement the actionPerformed method.

In this method, the `setDocument()` method of the ELJBean is used to set EditLive! for Java Swing <BODY> with the contents of the JTextArea. This method is invoked on the Swing Thread to ensure no threading problems can occur between the ELJBean and the rest of the application.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetDocument implements ActionListener {
    /** Buttons used to get html contents of JTextArea and set into EditLive */
    private JButton documentButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"));

    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);
        // add listener to button
        documentButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);
    }
}
```



```

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
     *
     * @param e ActionEvent sent by JButton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == documentButton) {
            try {
                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        editLiveBean.setDocument(source.getText());
                    }
                });
            } catch (Exception exception) {
                // TODO Auto-generated catch block
                exception.printStackTrace();
            }
        }
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetDocument();
    }
}

```

## See Also

- [<sourceEditor> Configuration Element](#)
- [getDocument\(\) method](#)

# Setting the Document in the Swing SDK Code

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.lang.reflect.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to populate the HTML
 * Document stored in EditLive!, at both load-time and run-time.
 *
 */
public class SetDocument extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton documentButton = new JButton("Set HTML Document");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 300, getClass().getResource
("sample_eljconfig.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public SetDocument() throws Exception {
        super("Tutorial - Setting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);
        // add listener to button
        documentButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // specify textarea content
        source.setText("<html><head><title>New Document Title</title></head><body><p>A New HTML Document
to load into <b>EditLive!</b></p></body></html>");
    }
}
```

```

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == documentButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    editLiveBean.setDocument(source.getText());
                }
            });
        } catch (Exception exception) {
            // TODO Auto-generated catch block
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new SetDocument();
}
}

```

# Setting the Body

- [Setting the Body in the Applet](#)
  - [Setting the Body in the Applet Code](#)
  - [Setting the Body in the Applet Tutorial](#)
- [Setting the Body in the Swing SDK](#)
  - [Setting the Body in the Swing SDK Tutorial](#)
  - [Setting the Body in the Swing SDK Code](#)

# Setting the Body in the Applet

For many developers integrating EditLive! into their application, the editor's purpose will be to allow users to generate extensively rich segments of HTML. Often the user isn't required to create a complete HTML document, but rather HTML that is then rendered out as a portion of a larger webpage.

To remove the complexity from having to deal with entire HTML documents when only snippets of HTML are desired, EditLive! provides the ability to set only the contents of the <BODY> element of its HTML Document.

This tutorial provides developers with the knowledge required to set only the <BODY> of the HTML Document stored in EditLive!. This tutorial depicts how the <BODY> of EditLive! can be set at both load-time or run-time.

## Tutorial

The [Setting the Body in the Applet Tutorial](#) provides a step-by-step walk-through on how to specify the <BODY> element of the HTML Document stored in EditLive!.

## Code

The complete code view for all the associated files in the Setting the Body in the Applet Tutorial is available [here](#).

# Setting the Body in the Applet Code

```
<!--
*****

setBody.html --

This tutorial shows developers how to populate the <BODY> of
a Document stored in EditLive!, at both load-time and run-time.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Setting the Body of the Editor's HTML Document - Tutorial</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Setting the Body of the Editor's HTML Document</h1>

    <form name="exampleForm">

      <p>This tutorial shows how to populate the &lt;BODY&gt; of the Document stored in
EditLive!</p>

      <p>This tutorial also shows how to extract the contents of a HTML field and store this
HTML in EditLive! Document &lt;BODY&gt;.</p>

      <!--
      The textarea used to load HTML content into the <body> of EditLive!'s HTML
document.
      -->
      <textarea name="bodyContents" cols="80" rows="5"><p>Text Area Contents.<br/><b>Note: <
/b>Ensure this text area contains correctly formatted <i>HTML</i></p></textarea>

      <!--
      The button for copying the content from the textarea to EditLive!
      -->
      <p>Pressing this button will copy the HTML contents of the above textarea into the &lt;
BODY&gt; of the Document stored in EditLive!<br/>
      <input type="button" value="Set <BODY> Contents" onclick="buttonPress()"></p>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");

        // Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.

        // Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
      </script>
    </body>
</html>
```

```

        // .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
        // at the this location.
        editlive.show();

        /** Function extracts the contents of the 'bodyContents' textarea and sets this
HTML into
        * the <body> field of the EditLive! HTML Document.
        */
        function buttonPress() {
            // Before sending HTML to the instance of EditLive!, this HTML must be
URL Encoded.
            // Javascript provides several URL Encoding methods, the best of which is
            // 'encodeURIComponent'
            editlive.setBody(encodeURIComponent(document.exampleForm.bodyContents.
value));
        }
    </script>

    </form>
</body>
</html>

```

# Setting the Body in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! in a Webpage

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! in a webpage.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.show();
    </script>
  </body>
</html>
```

Save this webpage as *setBody.html*.

### Step 2. Setting the <BODY> to Appear When the Editor Loads

The [body](#) load-time property is used to set the contents of the <BODY> element for the HTML Document stored in EditLive!.

For the purpose of this tutorial, we will load the following HTML into the editor:

<p>Original <i>HTML</i> loaded into EditLive!</p>

When specifying any HTML to insert into EditLive! for Java, this HTML must be URL Encoded. Javascript provides numerous URL Encoding methods (such as *escape* and *encodeURIComponent*). If using a server-side language to create the webpage where EditLive! is instantiated, Ephox advises using the URL encoding method of this language to encode the HTML to be used with either the load-time properties or run-time functions of EditLive!.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 3. Create a HTML Textarea



Add a HTML textarea to the webpage. The purpose of this textarea will be to allow users to write HTML content, which they can then send to EditLive! at any time.

```
<html>
  <body>
    <p><textarea id="bodyContents" cols="80" rows="5"></textarea></p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));

      editlivejava.show();
    </script>
  </body>
</html>
```

#### Step 4. Add a Button to Copy the Textarea Contents into EditLive!

In order to copy the contents of the textarea into EditLive!, a button should be created to call a Javascript method. This Javascript method will be written in the next step to perform the copying action.

```
<html>
  <body>
    <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
      <br/><input type="button" value="Set <BODY> Contents" onclick="buttonPress()"></p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));

      editlivejava.show();
    </script>
  </body>
</html>
```

#### Step 5. Create a Javascript Function to Set the <BODY> of EditLive! with the Textarea Contents

The [setBody Method](#) can be used to the <BODY> of EditLive!. You can directly reference the textarea's content by using the Javascript DOM methods.

As seen in Step 2, a URL encoding function is required when passing HTML to EditLive!.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Set <BODY> Contents" onclick="buttonPress()"><
/p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));

        editlivejava.show();

        function buttonPress() {
          editlivejava.setBody(encodeURIComponent(document.exampleForm.
bodyContents.value));
        }
      </script>
    </form>
  </body>
</html>
```

```
        }
      </script>
    </form>
  </body>
</html>
```

#### See Also

- [<sourceEditor>](#) Configuration Element
- [setBody Method](#)
- [getBody Method](#)

# Setting the Body in the Swing SDK

For many developers integrating EditLive! into their application, the editor's purpose will be to allow users to generate extensively rich segments of HTML. Often the user isn't required to create a complete HTML document, but rather HTML that is then rendered out as a portion of a larger webpage.

To remove the complexity from having to deal with entire HTML documents when only snippets of HTML are desired, EditLive! provides the ability to set only the contents of the <BODY> element of its HTML Document.

This tutorial provides developers with the knowledge required to set only the <BODY> of the HTML Document stored in EditLive!. This tutorial depicts how the <BODY> of EditLive! can be set at both load-time or run-time.

## Tutorial

The [Setting the Body in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to specify the <BODY> element of the HTML Document stored in EditLive!.

## Code

The complete code view for all the associated files in the Setting the Body in the Swing SDK Tutorial is available [here](#).

# Setting the Body in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! for Java Swing in a Webpage

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! for Java Swing in a `JFrame`.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetBody {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"));

    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
```

```

*
* @param args the command line arguments - ignored
*/
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetBody();
}
}

```

## Step 2. Setting the <BODY> to Appear When the Editor Loads

The [ELJBean](#) constructor can accept a HTML string to load into the Editor. This string can either be a complete HTML Document or simply the contents of the <BODY> element.

For the purpose of this tutorial, we will load the following HTML into the <BODY> of the EditLive! for Java Swing editor:

```
<p>Original <i>HTML</i> loaded into EditLive!</p>
```

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetBody {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}

```

```

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetBody();
}
}

```

### Step 3. Create a JTextArea and Add it to the JFrame

Add a [JTextArea](#) to the webpage. The purpose of this JTextArea will be to allow users to write HTML content, which they can then send to EditLive! for Java Swing at any time.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetBody {
    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
    "sample_eljconfig.xml"));

    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.

```

```

        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetBody();
    }
}

```

#### Step 4. Add a Button to Copy the JTextarea Contents into EditLive! for Java Swing

In order to copy the contents of the JTextArea into EditLive! for Java Swing, a [JButton](#) should be created to trigger the copying event. The listener for the button's click action will be added in the next step.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetBody {
    /** Buttons used to get html contents of JTextArea and set into EditLive */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
    "sample_eljconfig.xml"));

    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
    }
}

```

```

        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetBody();
    }
}

```

## Step 5. Extend the ActionListener and Add an ActionListener to the JButton

By implementing the [ActionListener](#) interface this class can listen to [ActionEvents](#). Add this class as an ActionListener to the JButton.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetBody implements ActionListener {
    /** Buttons used to get html contents of JTextArea and set into EditLive */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";
}

```



```

/** Base class for EditLive! for Java */
private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

public SetBody() throws Exception {
    super("Tutorial - Setting the <BODY> of the EditLive! Document");
    this.getContentPane().setLayout(new GridLayout(2, 1));

    SwingUtilities.invokeLaterAndWait(new Runnable() {
        public void run() {
            // initialize EditLive!
            editLiveBean.init();
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Create a JPanel to hold the button and the text area
    JPanel buttonAndText = new JPanel(new BorderLayout());

    // create a JPanel to hold the button
    JPanel buttonHolder = new JPanel(new FlowLayout());
    buttonHolder.add(bodyButton);
    // add listener to button
    bodyButton.addActionListener(this);

    // add button holding panel
    buttonAndText.add(buttonHolder, BorderLayout.NORTH);

    // create scrollable pane to hold text area
    JScrollPane textAreaHolder = new JScrollPane(source);
    buttonAndText.add(textAreaHolder);

    // add button and textarea to frame
    this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetBody();
}
}

```

## Step 6. Implementing the actionPerformed Method

Finally, to catch events fired by the JButton, implement the actionPerformed method.

In this method, the `setBody()` method of the ELJBean is used to set EditLive! for Java Swing <BODY> with the contents of the JTextArea. This method is invoked on the Swing Thread to ensure no threading problems can occur between the ELJBean and the rest of the application.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephex.editlive.*;

public class SetBody implements ActionListener {
    /** Buttons used to get html contents of JTextArea and set into EditLive! */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
("sample_eljconfig.xml")));

    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);
        // add listener to button
        bodyButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {

```

```

        System.exit(0);
    }
    });
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == bodyButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    editLiveBean.setBody(source.getText());
                }
            });
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetBody();
}
}

```

## See Also

- [<sourceEditor> Configuration Element](#)
- [getBody\(\) method](#)

# Setting the Body in the Swing SDK Code

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.lang.reflect.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to populate the <BODY> of
 * a Document stored in EditLive!, at both load-time and run-time.
 *
 */
public class SetBody extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 300, getClass().getResource
("sample_eljconfig.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public SetBody() throws Exception {
        super("Tutorial - Setting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);
        // add listener to button
        bodyButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // specify textarea content
        source.setText("<p>Text Area Contents.<br/><b>Note: </b>Ensure this text area contains correctly
formatted <i>HTML</i></p>");
    }
}
```

```

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == bodyButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    editLiveBean.setBody(source.getText());
                }
            });
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new SetBody();
}
}

```

# Getting the Document

- [Getting the Document in the Applet Overview](#)
  - [Getting the Document in the Applet Tutorial](#)
  - [Getting the Document in the Applet Code](#)
- [Getting the Document in the Swing SDK](#)
  - [Getting the Document in the Swing SDK Tutorial](#)
  - [Getting the Document in the Swing SDK Code](#)

# Getting the Document in the Applet Overview

This tutorial provides developers with the knowledge required to extract the HTML Document stored in EditLive!.

## Tutorial

The [Getting the Document in the Applet Tutorial](#) provides a step-by-step walk-through on how to extract the HTML Document stored in EditLive! at run-time.

When submitting a HTML form to a server, EditLive! will automatically create hidden form fields and populate these hidden form fields with the HTML contents of EditLive!. For more information, see the [Retrieving Content From EditLive!](#) article in the [Developer Guide](#).

## Code

The complete code view for all the associated files in the [Getting the Document in the Applet Tutorial](#) is available [here](#).

# Getting the Document in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Document in the Applet Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! in a Webpage and Set the Document

As shown in the [Setting the Document in the Applet Tutorial](#), create an instance of EditLive! in a webpage and set the Document.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

Save this webpage as *getDocument.html*.

### Step 2. Create a HTML Textarea and a Button.

As seen in the [Setting the Document in the Applet Tutorial](#), create a textarea and a button on the page. The purpose of the button in this tutorial will be to copy the contents of the HTML Document in EditLive! into the textarea.

```
<html>
  <body>
    <p><textarea id="documentContents" cols="80" rows="5"></textarea>
      <br/><input type="button" value="Get Document Contents" onclick="buttonPress()"></p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.setUseMathML(true);
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 3. Create a Javascript Function to Get the Document of EditLive! with the Textarea Contents

In order to extract the HTML Document stored in EditLive!, the [getDocument Method](#) is used.



The GetDocument run-time function requires a string parameter passed to be passed to it. This parameter is required to be the name of an existing Javascript method in the page. Once the [getDocument Method](#) is called, the Javascript property passed will then be called, passing the HTML document as a string parameter.

This tutorial contains a javascript function called *getEditLiveDocument*. When called, the string parameter passed to this function is then assigned to value of the textarea.

Hence, the [getDocument Method](#) passes the string '*getEditLiveDocument*', which then calls the *getEditLiveDocument* method, passing the HTML contents of EditLive! as the *src* parameter.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get Document Contents" onclick="buttonPress()"><
      /p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
  /script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
      editlivejava.show();

      /** Function extracts the contents of the EditLive! HTML Document
       * and displays this content in the textarea
       */
      function buttonPress() {
        // the parameter passed to the GetDocuement method is the callback
method the applet will call, passing
        // the contents of the HTML Document stored in EditLive!
        editlive.getBody('getEditLiveDocument');
      }

      function getEditLiveDocument(src) {
        document.exampleForm.documentContents.value = src;
      }
    </script>
  </form>
</body>
</html>
```

# Getting the Document in the Applet Code

```
<!--
*****

getDocument.html --

This tutorial shows developers how to extract the HTML
Document stored in EditLive! and display this HTML in
a webpage field.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Getting the Editor's HTML Document</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Getting the Editor's HTML Document</h1>

    <form name="exampleForm">
      <p>This tutorial shows how to extract the contents of the HTML Document in EditLive! and
display this HTML in a textarea on the webpage.</p>

      <!--
      The textarea used to display the HTML from EditLive!'s HTML document.
      -->
      <textarea id="documentContents" cols="80" rows="5">Pressing the Get HTML Contents button
will extract the HTML from EditLive! and place the HTML into this textarea.</textarea>

      <!--
      The button for copying the HTML Document from EditLive! and displaying this HTML
in the textarea
      -->
      <p><input type="button" value="Get HTML Contents" onclick="buttonPress()"></p>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");

        // Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.

        // Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive.setDocument(encodeURIComponent("<html><head><title>Default Document
Title</title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>"));

        // .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
        // at the this location.
editlive.show();

```

```
        /** Function extracts the contents of the EditLive! HTML Document
        * and displays this content in the textarea
        */
        function buttonPress() {
            editlive.getDocument('getEditLiveDocument');
        }

        function getEditLiveDocument(src) {
            document.exampleForm.documentContents.value = src;
        }
    </script>

    </form>
</body>
</html>
```

# Getting the Document in the Swing SDK

This tutorial provides developers with the knowledge required to extract the HTML Document stored in EditLive! for Java Swing.

## Tutorial

The [Getting the Document in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to extract the HTML Document stored in EditLive! for Java Swing at run-time.

## Code

The complete code view for all the associated files in the [Getting the Document in the Swing SDK Tutorial](#) is available [here](#).

# Getting the Document in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library

### Required Tutorials Completed

The following tutorials should be completed before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Document Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! for Java Swing in a Frame and Set the Body

As shown in the [Setting the Document](#) tutorial, create an instance of EditLive! for Java Swing in a JFrame and set the Document.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetDocument {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("sample_eljconfig.xml"));

    public GetDocument() throws Exception {
        super("Tutorial - Getting the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

```

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetDocument();
}
}

```

## Step 2. Create a JTextArea and a JButton and Add them to the JFrame

As seen in the Setting the Document tutorial, create a JTextArea and a JButton on the page. The purpose of the JButton in this tutorial will be to copy the contents of the HTML Document in EditLive! for Java Swing into the JTextArea.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetDocument {
    /** Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton documentButton = new JButton("Set Document");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public GetDocument() throws Exception {
        super("Tutorial - Getting the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);

        // add button holding panel

```

```

        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new GetDocument();
    }
}

```

### Step 3. Extend the ActionListener and Add an ActionListener to the JButton

By implementing the [ActionListener](#) interface this class can listen to [ActionEvents](#). Add this class as an ActionListener to the JButton.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetDocument implements ActionListener {
    /** Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton documentButton = new JButton("Set Document");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public GetDocument() throws Exception {
        super("Tutorial - Getting the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {

```

```

        public void run() {
            // initialize EditLive!
            editLiveBean.init();
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Create a JPanel to hold the button and the text area
    JPanel buttonAndText = new JPanel(new BorderLayout());

    // create a JPanel to hold the button
    JPanel buttonHolder = new JPanel(new FlowLayout());
    buttonHolder.add(documentButton);
    // add listener to button
    documentButton.addActionListener(this);

    // add button holding panel
    buttonAndText.add(buttonHolder, BorderLayout.NORTH);

    // create scrollable pane to hold text area
    JScrollPane textAreaHolder = new JScrollPane(source);
    buttonAndText.add(textAreaHolder);

    // add button and textarea to frame
    this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetDocument();
}
}

```

#### Step 4. Implementing the actionPerformed Method

Finally, to catch events fired by the JButton, implement the actionPerformed method.

In this method, the `getDocument()` method of the ELJBean is used to set the contents of the JTextArea with the EditLive! for Java Swing HTML Document. This method is invoked on the Swing Thread to ensure no threading problems can occur between the ELJBean and the rest of the application.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;

```



```

import javax.swing.*;
import com.ephox.editlive.*;

public class GetDocument implements ActionListener {
    /**
     * Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton documentButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public GetDocument() throws Exception {
        super("Tutorial - Getting the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);
        // add listener to button
        documentButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
    *
    * @param e ActionEvent sent by JButton
    */
}

```

```

*/
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == documentButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    source.setText(editLiveBean.getDocument());
                }
            });
        } catch (Exception exception) {
            // TODO Auto-generated catch block
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetDocument();
}
}

```

# Getting the Document in the Swing SDK Code

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.lang.reflect.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to extract the HTML
 * Document stored in EditLive! for Java Swing
 *
 */
public class GetDocument extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton documentButton = new JButton("Get HTML Document");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<html><head><title>Default Document Title</title><
/head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 300, getClass().getResource
("sample_eljconfig.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public GetDocument() throws Exception {
        super("Tutorial - Getting the HTML of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(documentButton);
        // add listener to button
        documentButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // specify textarea content
        source.setText("Pressing the above button will copy the contents of the HTML Document in
        EditLive! into this textarea.");
    }
}
```

```

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == documentButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    source.setText(editLiveBean.getDocument());
                }
            });
        } catch (Exception exception) {
            // TODO Auto-generated catch block
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new GetDocument();
}
}

```

# Getting the Body

- [Getting the Body in the Applet](#)
  - [Getting the Body in the Applet Tutorial](#)
  - [Getting the Body in the Applet Code](#)
- [Getting the Body in the Swing SDK](#)
  - [Getting the Body in the Swing SDK Tutorial](#)
  - [Getting the Body in the Swing SDK Code](#)

# Getting the Body in the Applet

For many developers integrating EditLive! into their application, the editor's purpose will be to allow users to generate extensively rich segments of HTML. Often the user isn't required to create a complete HTML document, but rather HTML that is rendered out as a portion of a larger webpage.

To remove the complexity from having to deal with entire HTML documents when only snippets of HTML are desired, EditLive! provides the ability to extract only the contents of the <BODY> element of its HTML Document.

This tutorial provides developers with the knowledge required to extract only the <BODY> of the HTML Document stored in EditLive! at run-time.

When submitting a HTML form to a server, EditLive! will automatically create hidden form fields and populate these hidden form fields with the HTML contents of EditLive!. For more information, see the [Retrieving Content From EditLive!](#) article in the [Developer Guide](#).

## Tutorial

The [Getting the Body in the Applet Tutorial](#) provides a step-by-step walk-through on how to extract the <BODY> element of the HTML Document stored in EditLive!.

## Code

The complete code view for all the associated files in the [Getting the Body in the Applet Tutorial](#) is available [here](#).

# Getting the Body in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Body in the Applet Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! in a Webpage and Set the Body

As shown in the [Setting the Body in the Applet Tutorial](#), create an instance of EditLive! in a webpage and set the <BODY>.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

Save this webpage as *getBody.html*.

### Step 2. Create a HTML Textarea and a Button.

As seen in the [Setting the Body in the Applet Tutorial](#), create a textarea and a button on the page. The purpose of the button in this tutorial will be to copy the contents of the <BODY> element in EditLive! into the textarea.

```
<html>
  <body>
    <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
      <br/><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"></p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive!<
/p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

### Step 3. Create a Javascript Function to Get the <BODY> of EditLive! with the Textarea Contents

In order to extract the <BODY> of the HTML Document stored in EditLive!, the [getBody Method](#) is used.

The `GetBody` run-time function requires a string parameter passed to it. This parameter is required to be the name of an existing Javascript method in the page. Once the `getBody Method` is called, the Javascript property passed will then be called, passing the `<BODY>` contents as a string parameter.

This tutorial contains a Javascript function called `getEditLiveBody`. When called, the string parameter passed to this function is then assigned to value of the textarea.

Hence, the `GetBody` property passes the string `'getEditLiveBody'`, which then calls the `getEditLiveBody` method, passing the `<BODY>` contents of `EditLive!` as the `src` parameter.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"><
    /p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
  /script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava
    /sample_eljconfig.xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
    EditLive!</p>"));
      editlivejava.show();

      /** Function extracts the contents of the <body> field of the EditLive! HTML
    Document
      * and displays this content in the textarea
      */
      function buttonPress() {
        // the parameter passed to the GetBody method is the callback method the
    applet will call, passing
        // the contents of the <BODY> attribute.
        editlive.getBody('getEditLiveBody');
      }

      function getEditLiveBody(src) {
        document.exampleForm.bodyContents.value = src;
      }
    </script>
  </form>
</body>
</html>
```

## See Also

- [<sourceEditor>](#) Configuration Element



# Getting the Body in the Applet Code

```
<!--
*****

getBody.html --

This tutorial shows developers how to extract the contents
of the <BODY> attribute for the HTML Document stored in
EditLive! and store this HTML in a webpage field.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Getting the Body of the Editor's HTML Document</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Getting the Body of the Editor's HTML Document</h1>

    <form name="exampleForm">

      <p>This tutorial shows how to extract the contents of the &lt;BODY&gt; attribute of the
HTML Document in EditLive! and display this HTML in a textarea on the webpage.</p>

      <!--
      The textarea used to display the HTML from the <body> of EditLive!'s HTML
document.
      -->
      <textarea id="bodyContents" cols="80" rows="5">Pressing the Get <BODY> Contents button
will extract the <BODY> from EditLive! and place the HTML into this textarea.</textarea>

      <!--
      The button for copying the <BODY> content from EditLive! and displaying this
HTML in the textarea
      -->
      <p><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"></p>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");

        // Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.
        // Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));

        // .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
        // at the this location.
      </script>
    </body>
</html>
```

```
editlive.show();

/** Function extracts the contents of the <body> field of the EditLive! HTML
Document
* and displays this content in the textarea
*/
function buttonPress() {
applet will call, passing
    // the parameter passed to the GetBody method is the callback method the
    // the contents of the <BODY> attribute.
    editlive.getBody('getEditLiveBody');
}

function getEditLiveBody(src) {
    document.exampleForm.bodyContents.value = src;
}
</script>

</form>
</body>
</html>
```

# Getting the Body in the Swing SDK

For many developers integrating EditLive! for Java Swing into their application, the editor's purpose will be to allow users to generate extensively rich segments of HTML. Often the user isn't required to create a complete HTML document, but rather HTML that is rendered out as a portion of a larger webpage.

To remove the complexity from having to deal with entire HTML documents when only snippets of HTML are desired, EditLive! for Java Swing provides the ability to extract only the contents of the <BODY> element of its HTML Document.

This tutorial provides developers with the knowledge required to extract only the <BODY> of the HTML Document stored in EditLive! for Java Swing at run-time.

## Tutorial

The [Getting the Body in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to extract the <BODY> element of the HTML Document stored in EditLive!.

## Code

The complete code view for all the associated files in the [Getting the Body in the Swing SDK Tutorial](#) is available [here](#).

# Getting the Body in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library

### Required Tutorials Completed

The following tutorials should be completed before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Body Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive! for Java Swing in a Frame and Set the Body

As shown in the [Setting the Body Tutorial](#), create an instance of EditLive! for Java Swing in a JFrame and set the <BODY>.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetBody {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
        "sample_eljconfig.xml"));

    public GetBody() throws Exception {
        super("Tutorial - Getting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

```

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetBody();
}
}

```

## Step 2. Create a JTextArea and a JButton and Add them to the JFrame

As seen in the [Setting the Body Tutorial](#), create a JTextArea and a JButton on the page. The purpose of the JButton in this tutorial will be to copy the contents of the <BODY> element in EditLive! for Java Swing into the JTextArea.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetBody {
    /** Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
    "sample_eljconfig.xml"));

    public GetBody() throws Exception {
        super("Tutorial - Getting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);
    }
}

```

```

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 620));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetBody();
}
}

```

### Step 3. Extend the ActionListener and Add an ActionListener to the JButton

By implementing the [ActionListener](#) interface this class can listen to [ActionEvents](#). Add this class as an ActionListener to the JButton.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class GetBody implements ActionListener {
    /** Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public GetBody() throws Exception {
        super("Tutorial - Getting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
            }
        });
    }
}

```

```

        editLiveBean.init();
    }
});

// Create a JPanel to hold the ELJBean
JPanel editorHolder = new JPanel(new FlowLayout());
editorHolder.add(editLiveBean);
// Add the JPanel to the JFrame
this.getContentPane().add(editorHolder);

// Create a JPanel to hold the button and the text area
JPanel buttonAndText = new JPanel(new BorderLayout());

// create a JPanel to hold the button
JPanel buttonHolder = new JPanel(new FlowLayout());
buttonHolder.add(bodyButton);
// add listener to button
bodyButton.addActionListener(this);

// add button holding panel
buttonAndText.add(buttonHolder, BorderLayout.NORTH);

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 620));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetBody();
}
}

```

## Step 4. Implementing the actionPerformed Method

Finally, to catch events fired by the JButton, implement the actionPerformed method.

In this method, the `getDocument()` method of the ELJBean is used to set the contents of the JTextArea with the EditLive! for Java Swing <BODY> HTML. This method is invoked on the Swing Thread to ensure no threading problems can occur between the ELJBean and the rest of the application.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

```

```

public class GetBody implements ActionListener {
    /**
     * Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("sample_eljconfig.xml"));

    public GetBody() throws Exception {
        super("Tutorial - Getting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);
        // add listener to button
        bodyButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // create scrollable pane to hold text area
        JScrollPane textAreaHolder = new JScrollPane(source);
        buttonAndText.add(textAreaHolder);

        // add button and textarea to frame
        this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
    *
    * @param e ActionEvent sent by JButton
    */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == bodyButton) {

```



```

        try {
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    source.setText(editLiveBean.getBody());
                }
            });
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetBody();
}
}

```

## See Also

- [<sourceEditor>](#) Configuration Element

# Getting the Body in the Swing SDK Code

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to extract the <BODY> of
 * a Document stored in EditLive! for Java Swing
 *
 */
public class GetBody extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton bodyButton = new JButton("Get <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 300, getClass().getResource(
    "sample_eljconfig.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public GetBody() throws Exception {
        super("Tutorial - Getting the <BODY> of the EditLive! Document");
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive!
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Create a JPanel to hold the button and the text area
        JPanel buttonAndText = new JPanel(new BorderLayout());

        // create a JPanel to hold the button
        JPanel buttonHolder = new JPanel(new FlowLayout());
        buttonHolder.add(bodyButton);
        // add listener to button
        bodyButton.addActionListener(this);

        // add button holding panel
        buttonAndText.add(buttonHolder, BorderLayout.NORTH);

        // specify textarea content
        source.setText("Pressing the above button will copy the contents of the <BODY> attribute in
    EditLive! into this textarea.");

        // create scrollable pane to hold text area
    
```

```

JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(710, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == bodyButton) {
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    source.setText(editLiveBean.getBody());
                }
            });
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new GetBody();
}
}

```

# Adding and Removing Menu or Toolbar Items

The purpose of this tutorial is to teach developers how to add and remove menu items or toolbar buttons from their EditLive! Configuration File.

The [Developer Guide](#) component of this SDK provides a complete list of all the available Menu items and Toolbar buttons for EditLive!. This list describes the function of each toolbar button and menu item, as well as specifying the unique XML Name attribute for each item. These unique Name attributes are used in the `<menuItem>` and `<toolbarButton>` Configuration File elements to specify the menu items and toolbar buttons, respectively, to appear in the instance of EditLive!.

## Tutorial

The [Adding and Removing Menu or Toolbar Items Tutorial](#) provides a step-by-step walk-through on how to adjust the menu items and toolbar buttons appearing on an instance of EditLive!.

## Code

The [complete code view](#) for all the associated files in the [Adding and Removing Menu or Toolbar Items Tutorial](#) is also available.

- [Adding and Removing Menu or Toolbar Items Tutorial](#)
- [Adding and Removing Menu or Toolbar Items Code](#)

# Adding and Removing Menu or Toolbar Items Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Creating and Editing Configuration Files](#)

## Adding and Removing Toolbar Buttons Using a Text Editor

### Step 1. Open the sample\_eljconfig.xml Configuration File

For your EditLive! SDK, open the *redistributables/editlivejava* directory in your SDK. Locate the *sample\_eljconfig.xml* Configuration File. This file is the default Configuration File provided by Tiny.

Open the *sample\_eljconfig.xml* Configuration File using a text editor.

### Step 2. Remove the Table Menu

EditLive! menu items are stored as `<menulitem>` elements. These `<menulitem>` elements are embedded in `<menu>` elements. `<menu>` elements define the menu name (such as File or Edit). Use the [Reference](#) section of this SDK to read up on the `<menu>` and `<menulitem>` Configuration File elements.

In this tutorial we will edit the *sample\_eljconfig.xml* file to:

- Remove all of the table related menu items and toolbar buttons.
- Add the Insert Horizontal Rule toolbar button and menu item.

Open the *sample\_eljconfig.xml* file using a text editor. Locate the `<menu name="ephox_tablemenu">` menu.

```
<menu name="ephox_tablemenu">
  <menulitem name="InsTable"/>
  <menulitem name="InsRowCol"/>
  <menulitem name="InsCell"/>
  <menuSeparator/>
  <menulitem name="DelRow"/>
  <menulitem name="DelCol"/>
  <menulitem name="DelCell"/>
  <menuSeparator/>
  <menulitem name="Split"/>
  <menulitem name="Merge"/>
  <menuSeparator/>
  <menulitem name="PropCell"/>
  <menulitem name="PropRow"/>
  <menulitem name="PropCol"/>
  <menulitem name="PropTable"/>
  <menuSeparator/>
  <menulitem name="Gridlines"/>
</menu>
```

Delete this entire element to remove the table menu and all its menu items. Save this Configuration File as *customized\_eljconfig.xml*.

### Step 3. Add a New Menu

Nest a new `<menu>` element with-in the `<menuBar>` element. Assign the value *Horizontal Rule* to the name attribute for the `<menu>` element.

```
<menuBar>
  ...
  <menu name="Horizontal Rule">
```

```
        </menu>
</menuBar>
```

#### Step 4. Add a New Menu Item to the Horizontal Rule Menu

The [Menu and Toolbar Item List Developer Guide](#) article specifies the unique value (when assigned to a `<menuItem>` or `<toolbarButton>` name attribute) used to create each menu item or toolbar button.

Within the `<menu name="Horizontal Rule">` element, nest a `<menuItem>` element with the value `HRule` assigned to its name attribute.

```
<menuBar>
    ...
    <menu name="Horizontal Rule">
        <menuItem name="HRule" />
    </menu>
</menuBar>
```

# Adding and Removing Menu or Toolbar Items Code

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

<!--

This file customizes and configures EditLive!.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFFFCC;
          }
        </style>
        -->
      </head>

      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
    -->
  </document>
</editlive>
```

```

-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    domain="LOCALHOST"
    key="6FFF-4DC5-EDF4-2486"
    licensee="For Evaluation Only"
    release="8.0"
    type="Evaluation License"
    productivityPack="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
  tabPlacement="bottom"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false"
  enableTrackChanges="false"
>
<!-- Define Custom Tags actions -->
<!--
<customTags>
  <doubleClickActions>
    <action.../>
  </doubleClickActions>
</customTags>
-->
<!-- Define additional symbols for the symbol dialog here -->
<!--
<symbols></symbols>
-->
</wysiwygEditor>
<!--

```



```

Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
  <!--
  Specify HTTP upload settings
  'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
  'href' is your server-side script for handling multipart-formdata uploads from ELJ
  The httpUploadData element specifies any additional fields to post with the image data
  -->
  <!--
  <httpUpload
    base="http://www.yourserver.com/userfiles/"
    href="http://www.yourserver.com/scripts/upload.jsp">
    <httpUploadData name="hello" data="world"/>
  </httpUpload>
  -->

  <images allowLocalImages="true" allowUserSpecified="true">
    <!--
    The list of images which appear in the Insert Image dialog.
    TIP: Dynamically generate this from your database or repository to achieve an easy image library.
    -->

    <imageList>
      <image name="EditLive!"
        description="EditLive! Logo"
        alt="EditLive! Logo"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
        title="EditLive!" />

      <image name="iMac"
        alt="iMac Computer"
        description="iMac Computer"
        title="iMac"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

      <image name="Apple Computer"
        alt="Apple Computer"
        title="Apple Computer"
        description="Picture of a new Apple Computer"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

      <image name="IBM Thinkpad"
        alt="IBM Thinkpad"
        border="0"
        title="IBM Thinkpad"
        description="Picture of a new IBM Thinkpad"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"
        />

    </imageList>
  </images>
  <multimedia>
    <types>
      <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
        <param name="movie" />

```

```

        <param name="quality" />
        <param name="bgcolor" />
    </type>
    <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
        <param name="autohref" />
        <param name="autoplay" />
        <param name="bgcolor" />
        <param name="cache" />
        <param name="controller" />
        <param name="correction" />
        <param name="dontflattenwhensaving" />
        <param name="enablejavascript" />
        <param name="endtime" />
        <param name="fov" />
        <param name="height" />
        <param name="href" />
        <param name="kioskmode" />
        <param name="loop" />
        <param name="movieid" />
        <param name="moviename" />
        <param name="node" />
        <param name="pan" />
        <param name="playeveryframe" />
        <param name="qtsrccchokespeed" />
        <param name="scale" />
        <param name="starttime" />
        <param name="target" />
        <param name="targetcache" />
        <param name="tilt" />
        <param name="urlsubstitute" />
        <param name="volume" />
    </type>
    <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
        <param name="animationAtStart" />
        <param name="autoStart" />
        <param name="showControls" />
        <param name="clickToPlay" />
        <param name="transparentAtStart" />
    </type>
    <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
        <param name="animationAtStart" />
        <param name="autoStart" />
        <param name="showControls" />
        <param name="clickToPlay" />
        <param name="transparentAtStart" />
    </type>
    <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
    <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
    <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
</types>
</multimedia>
</mediaSettings>

<hyperlinks>

    <hyperlinkList>
        <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
        <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
        <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
    </hyperlinkList>

    <mailtoList>
        <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
    </mailtoList>

</hyperlinks>

<!--

```

## Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's copyright in the appropriate place(s) within your application.

-->

```
<menuBar showAboutMenu="true">

  <menu name="ephox_filemenu">
    <menuItem name="New" />
    <menuItem name="Open" />
    <menuSeparator />
    <menuItem name="Save" />
    <menuItem name="SaveAs" />
    <menuSeparator />
    <menuItem name="Print" />
  </menu>

  <menu name="ephox_editmenu">
    <menuItem name="Undo" />
    <menuItem name="Redo" />
    <menuSeparator />
    <menuItem name="Cut" />
    <menuItem name="Copy" />
    <menuItem name="Paste" />
    <menuItem name="PasteSpecial" />
    <menuSeparator />
    <menuItem name="Select" />
    <menuItem name="SelectAll" />
    <menuSeparator />
    <menuItem name="Find" />
    <menuSeparator />
  </menu>

  <menu name="ephox_viewmenu">
    <menuItemGroup name="SourceView" />
    <menuSeparator />
    <menuItem name="Popout" />
    <menuSeparator />
    <menuItem name="showDocumentNavigator" />
    <menuSeparator />
    <menuItem name="ParagraphMarker" />
  </menu>

  <menu name="ephox_insertmenu">

    <menuItem name="HLink" />
    <menuItem name="Bookmark" />
    <menuItem name="RemoveHyperlink" />
    <menuSeparator />
    <menuItem name="ImageServer" />
    <menuItem name="InsertObject" />
    <menuSeparator />
    <menuItem name="Symbol" />
    <menuItem name="HRule" />
    <menuSeparator />
    <menuItem name="DateTime" />
    <menuSeparator />
    <menuItem name="insertcomment" />
  </menu>

  <menu name="ephox_formatmenu">
    <submenu name="Style" />
    <submenu name="Face" />
    <submenu name="Size" />
    <menuSeparator />
    <menuItem name="Bold" />
    <menuItem name="Italic" />
    <menuItem name="Underline" />
    <menuSeparator />
  </menu>
</menuBar>
```

```

    <menuItemGroup name="Align" />
    <menuSeparator />
    <menuItemGroup name="List" />
    <menuItem name="DecreaseIndent" />
    <menuItem name="IncreaseIndent" />
    <menuItem name="PropList" />
    <menuSeparator />
    <menuItemGroup name="Script" />
    <menuItem name="Strike" />
    <menuSeparator />
    <menuItem name="RemoveFormatting" />
    <menuItem name="FormatPainter" />
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling" />
  <menuItem name="BackgroundSpellChecking" />
  <menuItem name="thesaurus" />
  <menuSeparator />
  <menuItem name="Accessibility" />
  <menuSeparator />

  <menuItem name="WordCount" />
</menu>

<menu name="Horizontal Rule">
  <menuItem name="HRule" />
</menu>
<menu name="ephox_formmenu">
  <menuItem name="InsForm" />
  <menuSeparator />
  <menuItem name="InsTextField" />
  <menuItem name="InsPasswordField" />
  <menuItem name="InsHiddenField" />
  <menuItem name="InsFileField" />
  <menuItem name="InsButtonField" />
  <menuItem name="InsSubmitField" />
  <menuItem name="InsResetField" />
  <menuItem name="InsCheckboxField" />
  <menuItem name="InsRadioField" />
  <menuItem name="InsTextAreaField" />
  <menuItem name="InsSelectField" />
  <menuItem name="InsImageField" />
</menu>
<menu name="ephox_trackchangesmenu">
  <menuItem name="enabletrackchanges" />
  <menuSeparator />
  <menuItem name="acceptChange" />
  <menuItem name="rejectChange" />
  <menuSeparator />
  <menuItem name="previousChange" />
  <menuItem name="nextChange" />
  <menuSeparator />
  <menuItem name="acceptAllChanges" />
  <menuItem name="rejectAllChanges" />
  <menuSeparator />
  <menuItem name="showTrackChangesDialog" />
  <menuSeparator />
  <menuItem name="setUsername" />
</menu>
</menuBar>

<!--
Customize the EditLive! toolbars
-->
<toolbar>
  <toolbar name="Command">
    <toolbarButton name="Print" />
    <toolbarSeparator />
    <toolbarButton name="Spelling" />

```

```

<toolbarButton name="Find" />
<toolbarSeparator />
<toolbarButton name="Cut" />
<toolbarButton name="Copy" />
<toolbarButton name="Paste" />
<toolbarButton name="FormatPainter" />
<toolbarSeparator />
<toolbarButton name="Undo" />
<toolbarButton name="Redo" />
<toolbarSeparator />
<toolbarButton name="HLink" />
<toolbarButton name="ImageServer" />
<toolbarButton name="insertequation" />
<toolbarSeparator />
<toolbarButton name="enableTrackChanges" />
<toolbarButton name="acceptChange" />
<toolbarButton name="rejectchange" />
<toolbarButton name="previouschange" />
<toolbarButton name="nextchange" />
<toolbarSeparator />
<toolbarButton name="ParagraphMarker" />
<toolbarSeparator />
<toolbarButton name="Popout" />
</toolbar>

```

```
<toolbar name="Format">
```

```
<!--
```

Styles from any embedded or external stylesheets will also be automatically added to the Styles

drop-down

```
-->
```

```
<toolbarComboBox name="Style">
```

```

<comboBoxItem name="P" />
<comboBoxItem name="H1" />
<comboBoxItem name="H2" />
<comboBoxItem name="H3" />
<comboBoxItem name="H4" />
<comboBoxItem name="H5" />
<comboBoxItem name="H6" />

```

```
</toolbarComboBox>
```

```
<!--
```

You can remove the Font drop-down if you just want users to use Styles.

The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X

and Windows

To change the default font, change the embedded style sheet in the 'style' element above.

```
-->
```

```
<toolbarComboBox name="Face">
```

```

<comboBoxItem name="Arial" text="Arial" />
<comboBoxItem name="Arial Black" text="Arial Black" />
<comboBoxItem name="Arial Narrow" text="Arial Narrow" />
<comboBoxItem name="Comic Sans MS" text="Comic Sans MS" />
<comboBoxItem name="Courier New" text="Courier New" />
<comboBoxItem name="Georgia" text="Georgia" />
<comboBoxItem name="Impact" text="Impact" />
<comboBoxItem name="Times New Roman" text="Times New Roman" />
<comboBoxItem name="Trebuchet MS" text="Trebuchet MS" />
<comboBoxItem name="Verdana" text="Verdana" />

```

```
</toolbarComboBox>
```

```
<!--
```

Font Size drop-down

```
-->
```

```
<toolbarComboBox name="Size">
```

```

<comboBoxItem name="1" text="8pt" />
<comboBoxItem name="2" text="10pt" />
<comboBoxItem name="3" text="12pt" />
<comboBoxItem name="4" text="14pt" />
<comboBoxItem name="5" text="18pt" />
<comboBoxItem name="6" text="24pt" />
<comboBoxItem name="7" text="36pt" />

```

```
</toolbarComboBox>
```

```
<toolbarSeparator />
```

```
<toolbarButton name="Bold" />
```

```

        <toolbarButton name="Italic"/>
        <toolbarButton name="Underline"/>
        <toolbarSeparator/>
        <toolbarButton name="HRule"/>
        <toolbarSeparator/>
        <toolbarButtonGroup name="Align"/>
        <toolbarSeparator/>
        <toolbarButtonGroup name="List"/>
        <toolbarButton name="DecreaseIndent"/>
        <toolbarButton name="IncreaseIndent"/>
        <toolbarSeparator/>
        <toolbarButton name="HighlightColor"/>
        <toolbarButton name="Color"/>
    </toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo"/>
        <shrtMenuItem name="Redo"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Cut"/>
        <shrtMenuItem name="Copy"/>
        <shrtMenuItem name="Paste"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Select"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="acceptChange"/>
        <shrtMenuItem name="rejectChange"/>
        <shrtMenuItem name="nextchange"/>
        <shrtMenuItem name="previouschange"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Hyperlink"/>
        <shrtMenuItem name="RemoveHyperlink"/>
        <shrtMenuItem name="PropImage"/>
        <shrtMenuItem name="PropObject"/>
        <shrtMenuItem name="PropList"/>
        <shrtMenuItem name="PropHR"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Split"/>
        <shrtMenuItem name="Merge"/>
        <shrtMenuItem name="tableautofit"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="PropTable"/>
        <shrtMenuItem name="PropRow"/>
        <shrtMenuItem name="PropCol"/>
        <shrtMenuItem name="PropCell"/>

        <shrtMenuSeparator/>
        <shrtMenuItem name="synonyms"/>
        <shrtMenuItem name="EditTag"/>
    </shrtMenu>
</shortcutMenu>
</editlive>

```

# Specifying Character Set

- [Specifying Character Set in the Applet](#)
  - [Specifying Character Set in the Applet Tutorial](#)
  - [Specifying the Character Set in the Applet Code](#)
    - [charEncoding.html](#)
    - [charEncoding.xml](#)
- [Specifying Character Set in the Swing SDK](#)
  - [Specifying Character Set in the Swing SDK Tutorial](#)
  - [Specifying Character Set in the Swing SDK Code](#)
    - [charEncoding.java](#)
    - [charEncoding.xml \(Java Swing\)](#)

# Specifying Character Set in the Applet

EditLive! allows developers to specify the following editor properties:

- The character set used by the editor to decode HTML in EditLive! in order to display the HTML to the user.
- The character set used to encode the HTML extracted from EditLive! (using run-time properties such as the [getDocument Method](#)).

These separate character encoding specifications are set in two entirely different ways. This tutorial shows you how to set both of these character encoding specifications.

## Tutorial

The [Specifying Character Set in the Applet Tutorial](#) provides a step-by-step walk-through on how to specify both character encoding types for EditLive!.

## Code

The complete code views for all the associated files in the [Specifying Character Set in the Applet Tutorial](#) are available [here](#).



# Specifying Character Set in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Body in the Applet Tutorial](#)
- [Getting the Body in the Applet Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Specifying the Decoding Character Set

The decoding character set is used by EditLive! to display the HTML loaded into the editor and created by the user.

The decoding character set used by EditLive! is specified by the `<meta>` tag in either the HTML document loaded into EditLive! or the Configuration File used by EditLive!. For more information on specifying the decoding character set, see the Developer Guide section of this SDK.

For the purpose of this tutorial, we will be loading HTML fragments into EditLive! (i.e setting the `<BODY>` element of the HTML Document). Hence, to define the decoding character set, we'll need to specify this information in the Configuration File.

Open the `sampleeljconfig.xml` file packaged with EditLive! using a text editor. Locate the following line of code:

```
<!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->
```

Remove the `<!--` and `-->` characters wrapping to tag.

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

This will specify the UTF-8 character set for decoding the HTML content in EditLive!. Save the file as `charEncoding.xml`.

### Step 2. Create an Instance of EditLive! in a Webpage and Set the Body

As shown in the [Setting the Body in the Applet Tutorial](#), create an instance of EditLive! in a webpage and set the `<BODY>`. Ensure you load the `charEncoding.xml` file you previously created.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
      language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/charEncoding.xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
      editlivejava.show();
    </script>
  </body>
</html>
```

Save this webpage as `charEncoding.html`.

### Step 3. Create Controls and Code to Extract the <BODY> of EditLive! and Display the HTML in the Textarea

As seen in the [Getting the Body in the Applet Tutorial](#), perform the following:

- Create a HTML textarea.
- Create a HTML button.
- Wrap the contents of the page in a HTML form.
- Create a method for the button to call that will extract the <BODY> contents of EditLive! and copy this content into the textarea.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"><
      /p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
  /script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava
/charEncoding.xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 2</p><p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
      editlivejava.show();

      /** Function extracts the contents of the <body> field of the EditLive! HTML
Document
      * and displays this content in the textarea
      */
      function buttonPress() {
        // the parameter passed to the GetBody method is the callback method the
applet will call, passing
        // the contents of the <BODY> attribute.
        editlivejava.getBody('getEditLiveBody');
      }
      function getEditLiveBody(src) {
        document.exampleForm.bodyContents.value = src;
      }
    </script>
  </form>
</body>
</html>
```

### Step 4. Create Another Instance of EditLive!

To create another instance of EditLive! in the webpage. To do this, create a javascript variable and assign an instance of EditLive! to this variable by calling the [EditLiveJava constructor](#). You'll need to specify a different name in this constructor than the name specified in the previous call to the [EditLiveJava constructor](#). In the example below, the first instance of EditLive! specified the name *ELJApplet*. For our second instance of EditLive!, we'll specify the name *ELJApplet2*.

```
<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"><
      /p>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
  /script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava
/charEncoding.xml");
      editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 2</p><p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
      editlivejava.show();
```

```

Document
    /** Function extracts the contents of the <body> field of the EditLive! HTML
    * and displays this content in the textarea
    */
    function buttonPress() {
        // the parameter passed to the GetBody method is the callback method the
    applet will call, passing
        // the contents of the <BODY> attribute.
        editlivejava.getBody('getEditLiveBody');
    }

    var editlivejava2 = new EditLiveJava("ELJApplet2", "700", "400");

    function getEditLiveBody(src) {
        document.exampleForm.bodyContents.value = src;
    }
    </script>
    </form>
    </body>
</html>

```

Now you can apply the other load-time properties required to instantiate an instance of EditLive!.

```

<html>
    <body>
        <form name="exampleForm">
            <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
            <br/><input type="button" value="Get <BODY> Contents" onclick="buttonPress()"><
        /p>
        <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
    /script>
        <script>
            var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
            editlivejava.setConfigurationFile("../../redistributables/editlivejava
    /sample_eljconfig.xml");
            editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
    EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
            editlivejava.show();

            /** Function extracts the contents of the <body> field of the EditLive! HTML
            Document
            * and displays this content in the textarea
            */
            function buttonPress() {
                // the parameter passed to the GetBody method is the callback method the
            applet will call, passing
                // the contents of the <BODY> attribute.
                editlivejava.getBody('getEditLiveBody');
            }

            var editlivejava2 = new EditLiveJava("ELJApplet2", "700", "400");
            editlivejava2.setConfigurationFile("../../redistributables/editlivejava
    /charEncoding.xml");
            editlivejava2.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
    EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
            editlivejava2.show();

            function getEditLiveBody(src) {
                document.exampleForm.bodyContents.value = src;
            }
        </script>
        </form>
    </body>
</html>

```

Another HTML button and corresponding *onClick* method will also need to be generated to copy the contents of this new instance of EditLive! into the textarea.

```

<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get <BODY> Contents" onclick="button1Press()"><
      /p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
    /script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
        /sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
        EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
        editlivejava.show();

        /** Function extracts the contents of the <body> field of the EditLive! HTML
        Document
        * and displays this content in the textarea
        */
        function button1Press() {
          // the parameter passed to the GetBody method is the callback method the
          applet will call, passing
          // the contents of the <BODY> attribute.
          editlivejava.getBody('getEditLiveBody');
        }
      </script>
      <p><input type="button" value="Get <BODY> Contents for the Second Instance of EditLive!"
      onclick="button2Press()"></p>
      <script language="Javascript">
        var editlivejava2 = new EditLiveJava("ELJApplet2", "700", "400");
        editlivejava2.setConfigurationFile("../../redistributables/editlivejava
        /charEncoding.xml");
        editlivejava2.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
        EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
        editlivejava2.show();

        function button2Press() {
          // the parameter passed to the GetBody method is the callback method the
          applet will call, passing
          // the contents of the <BODY> attribute.
          editlivejava2.getBody('getEditLiveBody');
        }

        function getEditLiveBody(src) {
          document.exampleForm.bodyContents.value = src;
        }
      </script>
    </form>
  </body>
</html>

```

## Step 5. Specifying the Output Character Set

Finally, apply the [setOutputCharset Method](#) to the second instance of EditLive!, specifying the ASCII character set. When running this code, users will be able to see the differences in the HTML extracted from the first editor instance (using the default UTF-8 output character set) and the second editor instance (using the ASCII character set).

```

<html>
  <body>
    <form name="exampleForm">
      <p><textarea id="bodyContents" cols="80" rows="5"></textarea>
        <br/><input type="button" value="Get <BODY> Contents" onclick="button1Press()"><
      /p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
    /script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");

```

```

editlivejava.setConfigurationFile("../redistributables/editlivejava
/sample_eljconfig.xml");
editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
editlivejava.show();

/** Function extracts the contents of the <body> field of the EditLive! HTML
Document
* and displays this content in the textarea
*/
function button1Press() {
// the parameter passed to the GetBody method is the callback method the
applet will call, passing
// the contents of the <BODY> attribute.
editlivejava.getBody('getEditLiveBody');
}

</script>
<p><input type="button" value="Get <BODY> Contents for the Second Instance of EditLive!"
onclick="button2Press()"></p>
<script language="Javascript">
var editlivejava2 = new EditLiveJava("ELJApplet2", "700", "400");
editlivejava2.setConfigurationFile("../redistributables/editlivejava
/charEncoding.xml");
editlivejava2.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
editlivejava2.setOutputCharset("ASCII");
editlivejava2.show();

function button2Press() {
// the parameter passed to the GetBody method is the callback method the
applet will call, passing
// the contents of the <BODY> attribute.
editlivejava2.getBody('getEditLiveBody');
}

function getEditLiveBody(src) {
document.exampleForm.bodyContents.value = src;
}
</script>
</form>
</body>
</html>

```

#### See Also

- [sourceEditor](#) Configuration Element

# Specifying the Character Set in the Applet Code

- [charEncoding.html](#)
- [charEncoding.xml](#)

# charEncoding.html

```
<!--
*****

charEncoding.html --

This tutorial shows developers how to specify the encoding
used to render content in EditLive!, as well as specify the
encoding to use when extracting contents from EditLive!

This tutorial instantiates to separate instances of EditLive!
in the one webpage. One instance uses the default 'UTF-8' output
character encoding, the other uses 'ASCII' output character
encoding.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Specifying Character Encoding</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Specifying the Character Encoding Used by EditLive!</h1>

    <form name="exampleForm">

      <p>This tutorial shows how to specify the following types of character encoding:</p>

      <ul>
        <li>The character encoding used to render the HTML inside of EditLive!</li>
        <li>The character encoding used to extract the contents of EditLive!</li>
      </ul>

      <!--
      The textarea used to display the HTML from the <body> of EditLive!'s HTML
document.
      -->
      <textarea id="bodyContents" cols="80" rows="5">Use the respective buttons appearing in
this page to view the different encoding methods applied to each of the EditLive! instances.</textarea>

      <!--
      The button for copying the <BODY> content from EditLive! and displaying this
HTML in the textarea
      -->
      <p><input type="button" value="Get <BODY> Contents for the First Instance of EditLive!"
onclick="button1Press()"></p>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/charEncoding.
xml");
```

```

// Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.
// Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 1</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));
// .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
// at the this location.
editlive.show();

/** Function extracts the contents of the <body> field of the EditLive! HTML
Document
* and displays this content in the textarea
*/
function button1Press() {
// the parameter passed to the GetBody method is the callback method the
applet will call, passing
// the contents of the <BODY> attribute.
editlive.getBody('getEditLiveBody');
}
</script>

<!--
The button for copying the <BODY> content from EditLive! and displaying this
HTML in the textarea
-->
<p><input type="button" value="Get <BODY> Contents for the Second Instance of EditLive!"
onclick="button2Press()"></p>

<script language="Javascript">
// Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
var editlive2 = new EditLiveJava("ELApplet2", 700, 400);

// This sets a relative or absolute path to the XML configuration file to use
editlive2.setConfigurationFile("../..../redistributables/editlivejava/charEncoding.
xml");

// Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.
// Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive2.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive! Instance 2</p><p>Text with &#8220;Smart Quotes&#8221;</p>"));

// Setting the output character set to use 'ASCII' encoding.
// If the setOutoutCharset property is not specified, the default encoding
// for character output is 'UTF-8'.
editlive2.setOutputCharset("ASCII");

// .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
// at the this location.
editlive2.show();

/** Function extracts the contents of the <body> field of the EditLive! HTML
Document
* and displays this content in the textarea
*/
function button2Press() {
// the parameter passed to the GetBody method is the callback method the
applet will call, passing
// the contents of the <BODY> attribute.
editlive2.GetBody('getEditLiveBody');
}

/** Function utilised by both editor instances as a parameter passed to their
'GetBody' calls.
* this function creates a reference to the textarea on the page and inserts the
contents of

```



```
        * EditLive! into this textarea.
        */
        function getEditLiveBody(src) {
            document.exampleForm.bodyContents.value = src;
        }
    </script>
</form>
</body>
</html>
```

# charEncoding.xml

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

<!--

This file customizes and configures EditLive!.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings.

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFFFCC;
          }
        </style>
        -->
      </head>

      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
    -->
  </document>
</editlive>
```

```

-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    accountID="BB56B8DD47EF"
    activationURL="http://www.ephox.com/elregister/el2/activate.asp"
    domain="LOCALHOST"
    expiration="NEVER"
    forceActive="false"
    key="6FFF-9765-8052-7AA5"
    licensee="For Evaluation Only"
    product="EditLive! for Java"
    release="6.0"
    seats=""
    type="Evaluation License"
    eqEditor="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
  tabPlacement="bottom"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false"
  enableTrackChanges="false"
>
  <!-- Define Custom Tags actions -->
  <!--
  <customTags>
    <doubleClickActions>
      <action../>
    </doubleClickActions>
  </customTags>
  -->

```

```

    <!-- Define additional symbols for the symbol dialog here -->
    <!--
    <symbols></symbols>
    -->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
  <!--
  Specify HTTP upload settings
  'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
  'href' is your server-side script for handling multipart-formdata uploads from ELJ
  The httpUploadData element specifies any additional fields to post with the image data
  -->

  <httpUpload
    base="http://www.yourserver.com/userfiles/"
    href="http://www.yourserver.com/scripts/upload.jsp">

    <!--
    Specify any additional fields to post with the image data
    -->
    <!--<httpUploadData name="hello" data="world"/> -->

  </httpUpload>

  <images allowLocalImages="true" allowUserSpecified="true">
    <!--
    The list of images which appear in the Insert Image dialog.
    TIP: Dynamically generate this from your database or repository to achieve an easy image library.
    -->

    <imageList>
      <image name="Ephox EditLive!"
        description="Ephox EditLive! Logo"
        alt="Ephox EditLive! Logo"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
        title="Ephox EditLive!" />

      <image name="iMac"
        alt="iMac Computer"
        description="iMac Computer"
        title="iMac"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

      <image name="Apple Computer"
        alt="Apple Computer"
        title="Apple Computer"
        description="Picture of a new Apple Computer"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

      <image name="IBM Thinkpad"
        alt="IBM Thinkpad"
        border="0"

```

```

        title="IBM Thinkpad"
        description="Picture of a new IBM Thinkpad"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"
    />

</imageList>
</images>
<multimedia>
    <types>
        <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
            <param name="movie" />
            <param name="quality" />
            <param name="bgcolor" />
        </type>
        <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
            <param name="autohref" />
            <param name="autoplay" />
            <param name="bgcolor" />
            <param name="cache" />
            <param name="controller" />
            <param name="correction" />
            <param name="dontflattenwhensaving" />
            <param name="enablejavascript" />
            <param name="endtime" />
            <param name="fov" />
            <param name="height" />
            <param name="href" />
            <param name="kioskmode" />
            <param name="loop" />
            <param name="movieid" />
            <param name="moviename" />
            <param name="node" />
            <param name="pan" />
            <param name="playeveryframe" />
            <param name="qtsrccchokespeed" />
            <param name="scale" />
            <param name="starttime" />
            <param name="target" />
            <param name="targetcache" />
            <param name="tilt" />
            <param name="urlsubstitute" />
            <param name="volume" />
        </type>
        <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
        <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
        <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
    </types>
</multimedia>
</mediaSettings>

<hyperlinks>

    <hyperlinkList>

```

```

    <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
    <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
    <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
</hyperlinkList>

<mailtoList>
    <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
</mailtoList>

</hyperlinks>

<!--
Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

    <menu name="ephox_filemenu">
        <menuItem name="New"/>
        <menuItem name="Open"/>
        <menuSeparator/>
        <menuItem name="Save"/>
        <menuItem name="SaveAs"/>
        <menuSeparator/>
        <menuItem name="Print"/>
    </menu>

    <menu name="ephox_editmenu">
        <menuItem name="Undo"/>
        <menuItem name="Redo"/>
        <menuSeparator/>
        <menuItem name="Cut"/>
        <menuItem name="Copy"/>
        <menuItem name="Paste"/>
        <menuItem name="PasteSpecial"/>
        <menuSeparator/>
        <menuItem name="Select"/>
        <menuItem name="SelectAll"/>
        <menuSeparator/>
        <menuItem name="Find"/>
        <menuSeparator/>
    </menu>

    <menu name="ephox_viewmenu">
        <menuItemGroup name="SourceView"/>
        <menuSeparator/>
        <menuItem name="Popout"/>
        <menuSeparator/>
        <menuItem name="showDocumentNavigator"/>
        <menuSeparator/>
        <menuItem name="ParagraphMarker"/>
    </menu>

    <menu name="ephox_insertmenu">
        <menuItem name="HLink"/>
        <menuItem name="Bookmark"/>
        <menuItem name="RemoveHyperlink" />
        <menuSeparator/>
        <menuItem name="ImageServer"/>
        <menuItem name="InsertObject"/>
        <menuSeparator/>
        <menuItem name="Symbol"/>
        <menuItem name="HRule"/>
        <menuSeparator/>
        <menuItem name="DateTime"/>
        <menuSeparator/>
        <menuItem name="insertcomment"/>
    </menu>

```

```

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>

  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>

  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuItem name="tableautofit"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>
  <menuItem name="InsTextField"/>
  <menuItem name="InsPasswordField"/>
  <menuItem name="InsHiddenField"/>
  <menuItem name="InsFileField"/>
  <menuItem name="InsButtonField"/>
  <menuItem name="InsSubmitField"/>
  <menuItem name="InsResetField"/>
  <menuItem name="InsCheckboxField"/>
  <menuItem name="InsRadioField"/>
  <menuItem name="InsTextAreaField"/>
  <menuItem name="InsSelectField"/>
  <menuItem name="InsImageField"/>
</menu>

```

```

<menu name="ephox_trackchangesmenu">
  <menuItem name="enabletrackchanges" />
  <menuSeparator />
  <menuItem name="acceptChange" />
  <menuItem name="rejectChange" />
  <menuSeparator />
  <menuItem name="previousChange" />
  <menuItem name="nextChange" />
  <menuSeparator />
  <menuItem name="acceptAllChanges" />
  <menuItem name="rejectAllChanges" />
  <menuSeparator />
  <menuItem name="showTrackChangesDialog" />
  <menuSeparator />
  <menuItem name="setUsername" />
</menu>
</menuBar>

```

```

<!--
Customize the EditLive! toolbars
-->

```

```

<toolbars>
  <toolbar name="Command">

    <toolbarButton name="Print"/>
    <toolbarSeparator/>
    <toolbarButton name="Spelling"/>

    <toolbarButton name="Find"/>
    <toolbarSeparator/>
    <toolbarButton name="Cut"/>
    <toolbarButton name="Copy"/>
    <toolbarButton name="Paste"/>
    <toolbarButton name="FormatPainter" />
    <toolbarSeparator/>
    <toolbarButton name="Undo"/>
    <toolbarButton name="Redo"/>
    <toolbarSeparator/>
    <toolbarButton name="HLink"/>
    <toolbarButton name="ImageServer"/>
    <toolbarButton name="insertequation"/>
    <toolbarSeparator/>
    <toolbarButton name="InsTableWizard"/>
    <toolbarButton name="InsRow"/>
    <toolbarButton name="InsCol"/>
    <toolbarButton name="DelRow"/>
    <toolbarButton name="DelCol"/>

    <toolbarSeparator/>
    <toolbarButton name="enableTrackChanges"/>
    <toolbarButton name="acceptChange"/>
    <toolbarButton name="rejectchange"/>
    <toolbarButton name="previouschange"/>
    <toolbarButton name="nextchange"/>
    <toolbarSeparator/>
    <toolbarButton name="ParagraphMarker"/>
    <toolbarSeparator/>
    <toolbarButton name="Popout"/>
  </toolbar>

```

```

  <toolbar name="Format">
    <!--
    Styles from any embedded or external stylesheets will also be automatically added to the Styles
drop-down
    -->
    <toolbarComboBox name="Style">
      <comboBoxItem name="P"/>
      <comboBoxItem name="H1"/>
      <comboBoxItem name="H2"/>
      <comboBoxItem name="H3"/>
      <comboBoxItem name="H4"/>
    </toolbarComboBox>
  </toolbar>

```



```

        <comboBoxItem name="H5" />
        <comboBoxItem name="H6" />
    </toolbarComboBox>
    <!--
    You can remove the Font drop-down if you just want users to use Styles.
    The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
and Windows
    To change the default font, change the embedded style sheet in the 'style' element above.
    -->
    <toolbarComboBox name="Face">
        <comboBoxItem name="Arial" text="Arial" />
        <comboBoxItem name="Arial Black" text="Arial Black" />
        <comboBoxItem name="Arial Narrow" text="Arial Narrow" />
        <comboBoxItem name="Comic Sans MS" text="Comic Sans MS" />
        <comboBoxItem name="Courier New" text="Courier New" />
        <comboBoxItem name="Georgia" text="Georgia" />
        <comboBoxItem name="Impact" text="Impact" />
        <comboBoxItem name="Times New Roman" text="Times New Roman" />
        <comboBoxItem name="Trebuchet MS" text="Trebuchet MS" />
        <comboBoxItem name="Verdana" text="Verdana" />
    </toolbarComboBox>
    <!--
    Font Size drop-down
    -->
    <toolbarComboBox name="Size">
        <comboBoxItem name="1" text="8pt" />
        <comboBoxItem name="2" text="10pt" />
        <comboBoxItem name="3" text="12pt" />
        <comboBoxItem name="4" text="14pt" />
        <comboBoxItem name="5" text="18pt" />
        <comboBoxItem name="6" text="24pt" />
        <comboBoxItem name="7" text="36pt" />
    </toolbarComboBox>
    <toolbarSeparator />
    <toolbarButton name="Bold" />
    <toolbarButton name="Italic" />
    <toolbarButton name="Underline" />
    <toolbarSeparator />
    <toolbarButtonGroup name="Align" />
    <toolbarSeparator />
    <toolbarButtonGroup name="List" />
    <toolbarButton name="DecreaseIndent" />
    <toolbarButton name="IncreaseIndent" />
    <toolbarSeparator />
    <toolbarButton name="HighlightColor" />
    <toolbarButton name="Color" />
</toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo" />
        <shrtMenuItem name="Redo" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Cut" />
        <shrtMenuItem name="Copy" />
        <shrtMenuItem name="Paste" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Select" />
        <shrtMenuSeparator />
        <shrtMenuItem name="acceptChange" />
        <shrtMenuItem name="rejectChange" />
        <shrtMenuItem name="nextchange" />
        <shrtMenuItem name="previouschange" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Hyperlink" />
        <shrtMenuItem name="RemoveHyperlink" />
        <shrtMenuItem name="PropImage" />
    </shrtMenu>
</shortcutMenu>

```

```
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator />
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator />
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator />
    <shrtMenuItem name="synonyms" />
<shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# Specifying Character Set in the Swing SDK

EditLive! for Java Swing allows developers to specify the following editor properties:

- The character set used by the editor to decode HTML in EditLive! for Java Swing in order to display the HTML to the user.
- The character set used to encode the HTML extracted from EditLive! for Java Swing (using run-time properties such as [getDocument \(\)](#)).

These separate character encoding specifications are set in two entirely different ways. This tutorial shows you how to set both of these character encoding specifications.

## Tutorial

The [Specifying Character Set in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to specify both character encoding types for EditLive!.

## Code

The complete code views for all the associated files in the [Specifying Character Set in the Swing SDK Tutorial](#) are available [here](#).

# Specifying Character Set in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials should be completed before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Body Tutorial](#)
- [Getting the Body Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Specifying the Decoding Character Set

The decoding character set is used by EditLive! for Java Swing to decode the HTML loaded into the editor and created by the user.

The decoding character set used by EditLive! for Java Swing is specified by the <meta> tag in either the HTML document loaded into EditLive! for Java Swing or the Configuration File used by EditLive! for Java Swing. For more information on specifying the decoding character set, see the Developer Guide section of this SDK.

For the purpose of this tutorial, we will be loading HTML fragments into EditLive! for Java Swing (i.e setting the <BODY> element of the HTML Document). Hence, to define the decoding character set, we'll need to specify this information in the Configuration File.

Open the *sampleeljconfig.xml* file packaged with EditLive! for Java Swing using a text editor. Locate the following line of code:

```
<!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->
```

Remove the <!-- and --> characters wrapping to tag.

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

This will specify the UTF-8 character set for decoding the HTML content in EditLive! for Java Swing. Save the file as *charEncoding.xml*.

Step 2. Create an Instance of EditLive! for Java in a JFrame and Set the Body

As shown in the [Setting the Body in the Swing SDK Tutorial](#), create an instance of EditLive! for Java in a JFrame and set the <BODY>.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CharEncoding {
    /** html content to appear in the instance of EditLive! for Java Swing*/
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";

    /** Base class for EditLive! for Java Swing*/
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
("charEncoding.xml"), false);

    public CharEncoding() throws Exception {
        super("Tutorial - Specifying the Character Encoding");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
```

```

        public void run() {
            // initialize EditLive! instances
            editLiveBean.init();
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new GetBody();
}
}

```

Save this webpage as *CharEncoding.java*.

### Step 3. Create Controls and Code to Extract the <BODY> of EditLive! for Java Swing and Display the HTML in the Textarea

As seen in the [Getting the Body in the Swing SDK Tutorial](#), perform the following:

- Create a JTextarea
- Create a JButton
- Cause the class to implement an ActionListener. Assign the listener to the button. Cause the listener method to copy the contents of EditLive! for Java Swing into the JTextarea.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CharEncoding implements ActionListener {
    /** Buttons used to get html contents of EditLive and set into JTextArea */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<p>Original <i>HTML</i> loaded into EditLive!</p>";
}

```

```

/** Base class for EditLive! for Java */
private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("charEncoding.xml.xml"), false);

public CharEncoding() throws Exception {
    super("Tutorial - Specifying the Character Encoding");
    this.getContentPane().setLayout(new GridLayout(2, 1));

    SwingUtilities.invokeLaterAndWait(new Runnable() {
        public void run() {
            // initialize EditLive! instances
            editLiveBean.init();
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Create a JPanel to hold the button and the text area
    JPanel buttonAndText = new JPanel(new BorderLayout());

    // create a JPanel to hold the button
    JPanel buttonHolder = new JPanel(new FlowLayout());
    buttonHolder.add(bodyButton);
    // add listener to button
    bodyButton.addActionListener(this);

    // add button holding panel
    buttonAndText.add(buttonHolder, BorderLayout.NORTH);

    // create scrollable pane to hold text area
    JScrollPane textAreaHolder = new JScrollPane(source);
    buttonAndText.add(textAreaHolder);

    // add button and textarea to frame
    this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    try {
        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                if (e.getSource() == bodyButton) {
                    source.setText(editLiveBean.getBody());
                }
            }
        });
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
}

```

```

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new CharEncoding();
}
}

```

## Step 4. Create Another Instance of EditLive! for Java Swing

Create another instance of EditLive! for Java Swing in the JFrame. To do this, create another instance of the ELJBean class (you can use the same configuration file) and pack this bean into the frame.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

/** This tutorial shows developers how to specify the encoding for instances of EditLive! for Java Swing
 * and how to use several instances of EditLive! for Java Swing in the one JFrame.
 */
public class CharEncoding extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton bodyButton = new JButton("Set <BODY>");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML1 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 1<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";
    private static final String INITIAL_HTML2 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 2<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean1 = new ELJBean(INITIAL_HTML1, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);
    private ELJBean editLiveBean2 = new ELJBean(INITIAL_HTML2, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
 *
 */
    public CharEncoding() throws Exception {
        super("Tutorial - Specifying the Character Encoding");

        // Setting overall layout for frame
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive! instances
                editLiveBean1.init();
                editLiveBean2.init();
            }
        });
    }
}

```

```

// Creating panel for two EditLive! instances.
JPanel multiEditorHolder = new JPanel(new GridLayout(1, 2));

// Create a JPanel to hold the ELJBean
JPanel editorHolder1 = new JPanel(new FlowLayout());
editorHolder1.add(editLiveBean1);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder1);

// Create a JPanel to hold the ELJBean
JPanel editorHolder2 = new JPanel(new FlowLayout());
editorHolder2.add(editLiveBean2);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder2);

this.getContentPane().add(multiEditorHolder);

// Create a JPanel to hold the button and the text area
JPanel buttonAndText = new JPanel(new BorderLayout());

// create a JPanel to hold the button
JPanel buttonHolder = new JPanel(new FlowLayout());
buttonHolder.add(bodyButton);
// add listener to button
bodyButton.addActionListener(this);

// add button holding panel
buttonAndText.add(buttonHolder, BorderLayout.NORTH);

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(1210, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    try {
        SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
                if (e.getSource() == bodyButton) {
                    source.setText(editLiveBean.getBody());
                }
            }
        });
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {

```



```

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }

        new CharEncoding();
    }
}

```

## Step 5. Create Another Button to Extract the <BODY> and Adjust the ActionListener

A separate button now needs to be created to extract the <BODY> of the new instance of EditLive! for Java Swing. The actionPerformed method of the ActionListener also needs to be modified to detect which button was pressed in order to populate the JTextarea with the desired EditLive! for Java Swing <BODY>.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

/** This tutorial shows developers how to specify the encoding for instances of EditLive! for Java Swing
 * and how to use several instances of EditLive! for Java Swing in the one JFrame.
 */
public class CharEncoding extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton body1Button = new JButton("Get <BODY> for the above instance of EditLive!");
    private JButton body2Button = new JButton("Get <BODY> for the above instance of EditLive!");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML1 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 1<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";
    private static final String INITIAL_HTML2 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 2<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean1 = new ELJBean(INITIAL_HTML1, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);
    private ELJBean editLiveBean2 = new ELJBean(INITIAL_HTML2, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public CharEncoding() throws Exception {
        super("Tutorial - Specifying the Character Encoding");

        // Setting overall layout for frame
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive! instances
                editLiveBean1.init();
                editLiveBean2.init();
            }
        });
    }

    // Creating panel for two EditLive! instances.
    JPanel multiEditorHolder = new JPanel(new GridLayout(1, 2));

```

```

// Create a JPanel to hold the ELJBean
JPanel editorHolder1 = new JPanel(new FlowLayout());
editorHolder1.add(editLiveBean1);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder1);

// Create a JPanel to hold the ELJBean
JPanel editorHolder2 = new JPanel(new FlowLayout());
editorHolder2.add(editLiveBean2);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder2);

this.getContentPane().add(multiEditorHolder);

// Create a JPanel to hold the buttons and the text area
JPanel buttonsAndText = new JPanel(new BorderLayout());

// Creating a panel for the two buttons
JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
buttonsPanel.add(body1Button);
// add listener to button
body1Button.addActionListener(this);
buttonsPanel.add(body2Button);
// add listener to button
body2Button.addActionListener(this);

// add button holding panel
buttonsAndText.add(buttonsPanel, BorderLayout.NORTH);

// specify textarea content
source.setText("Pressing the above button will copy the contents of the <BODY> attribute in
EditLive! into this textarea.");

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonsAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonsAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(1210, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    try {
        SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
                if (e.getSource() == body1Button) {
                    source.setText(editLiveBean1.getBody());
                }
                if (e.getSource() == body2Button) {
                    source.setText(editLiveBean2.getBody());
                }
            }
        });
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

```

```

    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }

        new CharEncoding();
    }
}

```

## Step 6. Specifying Output Character Set

Finally, apply the `setOutputCharset` method to the second instance of `EditLive!` for Java Swing, specifying the ASCII character set. When running this code, users will be able to see the differences in the HTML extracted from the first editor instance (using the default UTF-8 output character set) and the second editor instance (using the ASCII character set).

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

/** This tutorial shows developers how to specify the encoding for instances of EditLive! for Java Swing
 * and how to use several instances of EditLive! for Java Swing in the one JFrame.
 */
public class CharEncoding extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton body1Button = new JButton("Get <BODY> for the above instance of EditLive!");
    private JButton body2Button = new JButton("Get <BODY> for the above instance of EditLive!");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML1 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 1<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";
    private static final String INITIAL_HTML2 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 2<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean1 = new ELJBean(INITIAL_HTML1, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);
    private ELJBean editLiveBean2 = new ELJBean(INITIAL_HTML2, "", 600, 300, getClass().getResource
("charEncoding.xml.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
     *
     */
    public CharEncoding() throws Exception {
        super("Tutorial - Specifying the Character Encoding");

        // Setting overall layout for frame
        this.getContentPane().setLayout(new GridLayout(2, 1));

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive! instances
                editLiveBean1.init();
            }
        });
    }
}

```

```

        editLiveBean2.init();

        // Setting the output character set to use 'ASCII' encoding.
        // If the setOutoutCharset property is not specified, the default encoding
        // for character output is 'UTF-8'.
        editLiveBean2.setOutputCharacterSet("ASCII");
    }
});

// Creating panel for two EditLive! instances.
JPanel multiEditorHolder = new JPanel(new GridLayout(1, 2));

// Create a JPanel to hold the ELJBean
JPanel editorHolder1 = new JPanel(new FlowLayout());
editorHolder1.add(editLiveBean1);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder1);

// Create a JPanel to hold the ELJBean
JPanel editorHolder2 = new JPanel(new FlowLayout());
editorHolder2.add(editLiveBean2);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder2);

this.getContentPane().add(multiEditorHolder);

// Create a JPanel to hold the buttons and the text area
JPanel buttonsAndText = new JPanel(new BorderLayout());

// Creating a panel for the two buttons
JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
buttonsPanel.add(body1Button);
// add listener to button
body1Button.addActionListener(this);
buttonsPanel.add(body2Button);
// add listener to button
body2Button.addActionListener(this);

// add button holding panel
buttonsAndText.add(buttonsPanel, BorderLayout.NORTH);

// specify textarea content
source.setText("Pressing the above button will copy the contents of the <BODY> attribute in
EditLive! into this textarea.");

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonsAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonsAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(1210, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == body1Button) {

```

```

        source.setText(editLiveBean1.getBody());
    }
    if (e.getSource() == body2Button) {
        source.setText(editLiveBean2.getBody());
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new CharEncoding();
}
}

```

## See Also

- [<sourceEditor>](#) Configuration Element

# Specifying Character Set in the Swing SDK Code

- [charEncoding.java](#)
- [charEncoding.xml \(Java Swing\)](#)

# charEncoding.java

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** This tutorial shows developers how to specify the encoding for instances of EditLive! for Java Swing
 *     and how to use several instances of EditLive! for Java Swing in the one JFrame.
 */
public class CharEncoding extends JFrame implements ActionListener {
    /** Buttons used to get html contents of EditLive! and copy to JTextArea source */
    private JButton body1Button = new JButton("Get <BODY> for the above instance of EditLive!");
    private JButton body2Button = new JButton("Get <BODY> for the above instance of EditLive!");

    /** Allows users to enter html content to insert into EditLive!, or holds a copy of the html contents of
    EditLive! */
    private JTextArea source = new JTextArea(10, 30);

    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML1 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 1<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";
    private static final String INITIAL_HTML2 = "<p>Original <i>HTML</i> loaded into EditLive! Instance 2<
/p><p>Text with &#8220;Smart Quotes&#8221;</p>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean1 = new ELJBean(INITIAL_HTML1, "", 600, 300, getClass().getResource
("charEncoding.xml"), false);
    private ELJBean editLiveBean2 = new ELJBean(INITIAL_HTML2, "", 600, 300, getClass().getResource
("charEncoding.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public CharEncoding() throws Exception {
        super("Tutorial - Specifying the Character Encoding");

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive! instances
                editLiveBean1.init();
                editLiveBean2.init();

                // Setting the output character set to use 'ASCII' encoding.
                // If the setOutoutCharset property is not specified, the default encoding
                // for character output is 'UTF-8'.
                editLiveBean2.setOutputCharacterSet("ASCII");
            }
        });

        // Setting overall layout for frame
        this.getContentPane().setLayout(new GridLayout(2, 1));

        // Creating panel for two EditLive! instances.
        JPanel multiEditorHolder = new JPanel(new GridLayout(1, 2));

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder1 = new JPanel(new FlowLayout());
        editorHolder1.add(editLiveBean1);
        // Add the JPanel to the JFrame
    }
}
```

```

multiEditorHolder.add(editorHolder1);

// Create a JPanel to hold the ELJBean
JPanel editorHolder2 = new JPanel(new FlowLayout());
editorHolder2.add(editLiveBean2);
// Add the JPanel to the JFrame
multiEditorHolder.add(editorHolder2);

this.getContentPane().add(multiEditorHolder);

// Create a JPanel to hold the buttons and the text area
JPanel buttonsAndText = new JPanel(new BorderLayout());

// Creating a panel for the two buttons
JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
buttonsPanel.add(body1Button);
// add listener to button
body1Button.addActionListener(this);
buttonsPanel.add(body2Button);
// add listener to button
body2Button.addActionListener(this);

// add button holding panel
buttonsAndText.add(buttonsPanel, BorderLayout.NORTH);

// specify textarea content
source.setText("Pressing the above button will copy the contents of the <BODY> attribute in
EditLive! into this textarea.");

// create scrollable pane to hold text area
JScrollPane textAreaHolder = new JScrollPane(source);
buttonsAndText.add(textAreaHolder);

// add button and textarea to frame
this.getContentPane().add(buttonsAndText, BorderLayout.CENTER);

// Display the JFrame.
this.setSize(new Dimension(1210, 640));
this.setVisible(true);

// Adding a listener to detect if the JFrame is closing, to close the application if needed.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

/** ActionListener for JButtons on the JFrame
 *
 * @param e ActionEvent sent by JButton
 */
public void actionPerformed(final ActionEvent e) {
    try {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                if (e.getSource() == body1Button) {
                    source.setText(editLiveBean1.getBody());
                }
                if (e.getSource() == body2Button) {
                    source.setText(editLiveBean2.getBody());
                }
            }
        });
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}

/** Sets up the application and begins its execution
 *

```



```
* @param args the command line arguments - ignored
*/
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new CharEncoding();
}
}
```

# charEncoding.xml (Java Swing)

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

<!--

This file customizes and configures EditLive! for Java Swing.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFFFCC;
          }
        </style>
        -->
      </head>

      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
    -->
  </document>
</editlive>
```

```

-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    accountID="BB56B8DD47EF"
    activationURL="http://www.ephox.com/elregister/el2/activate.asp"
    domain="LOCALHOST"
    expiration="NEVER"
    forceActive="false"
    key="6FFF-9765-8052-7AA5"
    licensee="For Evaluation Only"
    product="EditLive! for Java Swing"
    release="6.0"
    seats=""
    type="Evaluation License"
    eqEditor="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
  tabPlacement="bottom"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false"
  enableTrackChanges="false"
>
  <!-- Define Custom Tags actions -->
  <!--
  <customTags>
    <doubleClickActions>
      <action../>
    </doubleClickActions>
  </customTags>
  -->

```

```

    <!-- Define additional symbols for the symbol dialog here -->
    <!--
    <symbols></symbols>
    -->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
  <!--
  Specify HTTP upload settings
  'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
  'href' is your server-side script for handling multipart-formdata uploads from ELJ
  The httpUploadData element specifies any additional fields to post with the image data
  -->

  <httpUpload
    base="http://www.yourserver.com/userfiles/"
    href="http://www.yourserver.com/scripts/upload.jsp">

    <!--
    Specify any additional fields to post with the image data
    -->
    <!--<httpUploadData name="hello" data="world"/> -->

  </httpUpload>

  <images allowLocalImages="true" allowUserSpecified="true">
    <!--
    The list of images which appear in the Insert Image dialog.
    TIP: Dynamically generate this from your database or repository to achieve an easy image library.
    -->

    <imageList>
      <image name="Ephox EditLive! for Java Swing"
        description="Ephox EditLive! for Java Swing Logo"
        alt="Ephox EditLive! for Java Swing Logo"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
        title="Ephox EditLive! for Java Swing" />

      <image name="iMac"
        alt="iMac Computer"
        description="iMac Computer"
        title="iMac"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

      <image name="Apple Computer"
        alt="Apple Computer"
        title="Apple Computer"
        description="Picture of a new Apple Computer"
        border="0"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

      <image name="IBM Thinkpad"
        alt="IBM Thinkpad"
        border="0"

```

```

        title="IBM Thinkpad"
        description="Picture of a new IBM Thinkpad"
        src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"
    />

</imageList>
</images>
<multimedia>
    <types>
        <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
            <param name="movie" />
            <param name="quality" />
            <param name="bgcolor" />
        </type>
        <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
            <param name="autohref" />
            <param name="autoplay" />
            <param name="bgcolor" />
            <param name="cache" />
            <param name="controller" />
            <param name="correction" />
            <param name="dontflattenwhensaving" />
            <param name="enablejavascript" />
            <param name="endtime" />
            <param name="fov" />
            <param name="height" />
            <param name="href" />
            <param name="kioskmode" />
            <param name="loop" />
            <param name="movieid" />
            <param name="moviename" />
            <param name="node" />
            <param name="pan" />
            <param name="playeveryframe" />
            <param name="qtsrccchokespeed" />
            <param name="scale" />
            <param name="starttime" />
            <param name="target" />
            <param name="targetcache" />
            <param name="tilt" />
            <param name="urlsubstitute" />
            <param name="volume" />
        </type>
        <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
        <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
        <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
    </types>
</multimedia>
</mediaSettings>

<hyperlinks>

    <hyperlinkList>

```

```

    <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
    <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
    <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
</hyperlinkList>

<mailtoList>
    <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
</mailtoList>

</hyperlinks>

<!--
Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

    <menu name="ephox_filemenu">
        <menuItem name="New"/>
        <menuItem name="Open"/>
        <menuSeparator/>
        <menuItem name="Save"/>
        <menuItem name="SaveAs"/>
        <menuSeparator/>
        <menuItem name="Print"/>
    </menu>

    <menu name="ephox_editmenu">
        <menuItem name="Undo"/>
        <menuItem name="Redo"/>
        <menuSeparator/>
        <menuItem name="Cut"/>
        <menuItem name="Copy"/>
        <menuItem name="Paste"/>
        <menuItem name="PasteSpecial"/>
        <menuSeparator/>
        <menuItem name="Select"/>
        <menuItem name="SelectAll"/>
        <menuSeparator/>
        <menuItem name="Find"/>
        <menuSeparator/>
    </menu>

    <menu name="ephox_viewmenu">
        <menuItemGroup name="SourceView"/>
        <menuSeparator/>
        <menuItem name="Popout"/>
        <menuSeparator/>
        <menuItem name="showDocumentNavigator"/>
        <menuSeparator/>
        <menuItem name="ParagraphMarker"/>
    </menu>

    <menu name="ephox_insertmenu">
        <menuItem name="HLink"/>
        <menuItem name="Bookmark"/>
        <menuItem name="RemoveHyperlink" />
        <menuSeparator/>
        <menuItem name="ImageServer"/>
        <menuItem name="InsertObject"/>
        <menuSeparator/>
        <menuItem name="Symbol"/>
        <menuItem name="HRule"/>
        <menuSeparator/>
        <menuItem name="DateTime"/>
        <menuSeparator/>
        <menuItem name="insertcomment"/>
    </menu>

```

```

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>

  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>

  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuItem name="tableautofit"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>
  <menuItem name="InsTextField"/>
  <menuItem name="InsPasswordField"/>
  <menuItem name="InsHiddenField"/>
  <menuItem name="InsFileField"/>
  <menuItem name="InsButtonField"/>
  <menuItem name="InsSubmitField"/>
  <menuItem name="InsResetField"/>
  <menuItem name="InsCheckboxField"/>
  <menuItem name="InsRadioField"/>
  <menuItem name="InsTextAreaField"/>
  <menuItem name="InsSelectField"/>
  <menuItem name="InsImageField"/>
</menu>

```

```

<menu name="ephox_trackchangesmenu">
  <menuItem name="enabletrackchanges" />
  <menuSeparator />
  <menuItem name="acceptChange" />
  <menuItem name="rejectChange" />
  <menuSeparator />
  <menuItem name="previousChange" />
  <menuItem name="nextChange" />
  <menuSeparator />
  <menuItem name="acceptAllChanges" />
  <menuItem name="rejectAllChanges" />
  <menuSeparator />
  <menuItem name="showTrackChangesDialog" />
  <menuSeparator />
  <menuItem name="setUsername" />
</menu>
</menuBar>

```

```

<!--
Customize the EditLive! toolbars
-->

```

```

<toolbars>
  <toolbar name="Command">

    <toolbarButton name="Print"/>
    <toolbarSeparator/>
    <toolbarButton name="Spelling"/>

    <toolbarButton name="Find"/>
    <toolbarSeparator/>
    <toolbarButton name="Cut"/>
    <toolbarButton name="Copy"/>
    <toolbarButton name="Paste"/>
    <toolbarButton name="FormatPainter" />
    <toolbarSeparator/>
    <toolbarButton name="Undo"/>
    <toolbarButton name="Redo"/>
    <toolbarSeparator/>
    <toolbarButton name="HLink"/>
    <toolbarButton name="ImageServer"/>
    <toolbarButton name="insertequation"/>
    <toolbarSeparator/>
    <toolbarButton name="InsTableWizard"/>
    <toolbarButton name="InsRow"/>
    <toolbarButton name="InsCol"/>
    <toolbarButton name="DelRow"/>
    <toolbarButton name="DelCol"/>

    <toolbarSeparator/>
    <toolbarButton name="enableTrackChanges"/>
    <toolbarButton name="acceptChange"/>
    <toolbarButton name="rejectchange"/>
    <toolbarButton name="previouschange"/>
    <toolbarButton name="nextchange"/>
    <toolbarSeparator/>
    <toolbarButton name="ParagraphMarker"/>
    <toolbarSeparator/>
    <toolbarButton name="Popout"/>
  </toolbar>

```

```

  <toolbar name="Format">
    <!--
drop-down
    Styles from any embedded or external stylesheets will also be automatically added to the Styles
    -->
    <toolbarComboBox name="Style">
      <comboBoxItem name="P"/>
      <comboBoxItem name="H1"/>
      <comboBoxItem name="H2"/>
      <comboBoxItem name="H3"/>
      <comboBoxItem name="H4"/>
    </toolbarComboBox>
  </toolbar>

```



```

        <comboBoxItem name="H5"/>
        <comboBoxItem name="H6"/>
    </toolbarComboBox>
    <!--
    You can remove the Font drop-down if you just want users to use Styles.
    The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
and Windows
    To change the default font, change the embedded style sheet in the 'style' element above.
    -->
    <toolbarComboBox name="Face">
        <comboBoxItem name="Arial" text="Arial"/>
        <comboBoxItem name="Arial Black" text="Arial Black"/>
        <comboBoxItem name="Arial Narrow" text="Arial Narrow"/>
        <comboBoxItem name="Comic Sans MS" text="Comic Sans MS"/>
        <comboBoxItem name="Courier New" text="Courier New"/>
        <comboBoxItem name="Georgia" text="Georgia"/>
        <comboBoxItem name="Impact" text="Impact"/>
        <comboBoxItem name="Times New Roman" text="Times New Roman"/>
        <comboBoxItem name="Trebuchet MS" text="Trebuchet MS"/>
        <comboBoxItem name="Verdana" text="Verdana"/>
    </toolbarComboBox>
    <!--
    Font Size drop-down
    -->
    <toolbarComboBox name="Size">
        <comboBoxItem name="1" text="8pt"/>
        <comboBoxItem name="2" text="10pt"/>
        <comboBoxItem name="3" text="12pt"/>
        <comboBoxItem name="4" text="14pt"/>
        <comboBoxItem name="5" text="18pt"/>
        <comboBoxItem name="6" text="24pt"/>
        <comboBoxItem name="7" text="36pt"/>
    </toolbarComboBox>
    <toolbarSeparator/>
    <toolbarButton name="Bold"/>
    <toolbarButton name="Italic"/>
    <toolbarButton name="Underline"/>
    <toolbarSeparator/>
    <toolbarButtonGroup name="Align"/>
    <toolbarSeparator/>
    <toolbarButtonGroup name="List"/>
    <toolbarButton name="DecreaseIndent"/>
    <toolbarButton name="IncreaseIndent"/>
    <toolbarSeparator/>
    <toolbarButton name="HighlightColor"/>
    <toolbarButton name="Color"/>
    </toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo"/>
        <shrtMenuItem name="Redo"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Cut"/>
        <shrtMenuItem name="Copy"/>
        <shrtMenuItem name="Paste"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Select"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="acceptChange"/>
        <shrtMenuItem name="rejectChange"/>
        <shrtMenuItem name="nextchange"/>
        <shrtMenuItem name="previouschange"/>
        <shrtMenuSeparator/>
        <shrtMenuItem name="Hyperlink"/>
        <shrtMenuItem name="RemoveHyperlink"/>
        <shrtMenuItem name="PropImage"/>
    </shrtMenu>
</shortcutMenu>

```

```
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator />
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator />
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator />
    <shrtMenuItem name="synonyms" />
<shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# Setting CSS

- [Setting CSS in the Applet](#)
  - [Setting CSS in the Applet Tutorial](#)
  - [Setting CSS in the Applet Code](#)
- [Setting CSS in the Swing SDK](#)
  - [Setting CSS in the Swing SDK Tutorial](#)
  - [Setting CSS in the Swing SDK Code](#)

# Setting CSS in the Applet

EditLive! allows developers to specify the CSS for rendering the HTML content created within the editor. For a complete list of the methods available for specifying CSS for use with EditLive!, see the [Using CSS in the Applet](#) article in the [Developer Guide](#) section of this SDK.

This tutorial provides developers with the knowledge required to load CSS into EditLive! under the following circumstances:

- When CSS is stored in an external .css file, and
- When CSS is dynamically generated into the HTML page from a server-side script.

## Tutorial

The [Setting CSS in the Applet Tutorial](#) provides a step-by-step walk-through on how to load CSS into EditLive! from either an external CSS file or as a defined string of CSS.

## Code

The complete code view for all the associated files in the [Setting CSS in the Applet Tutorial](#) is available [here](#).

# Setting CSS in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Create a Reference to the External CSS File in the EditLive! Configuration File

To specify an external CSS file for use with EditLive!, you'll need to make changes to your EditLive! Configuration File. As outlined in the [Creating and Editing Configuration Files Tutorial](#), this can be done using a text editor.

#### Using a Text Editor to Specify an External CSS File

Open the *sampleeljconfig.xml* file packaged with EditLive! using a text editor. Locate the following line of code:

```
<!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->
```

Remove the `<!--` and `-->` characters wrapping to tag.

```
<link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/>
```

Change the `href` attribute to reference the *main.css* file packaged with this SDK. The URL for the location of the *main.css* can either be absolute or relative to the webpage where EditLive! is instantiated from.

```
<link rel="stylesheet" href="main.css" type="text/css"/>
```

Save this configuration file as *styles\_config.xml*.

### Step 2. Create an Instance of EditLive! for Java in a Webpage

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! in a webpage.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/styles_config.
xml");
      editlivejava.show();
    </script>
  </body>
</html>
```

Use the [setConfigurationFile Method](#) to specify the *styles\_config.xml* file.

Save this webpage as *setStyles.html*

### Step 3. Specify a String of CSS Using the Styles Load Time Property

Many web-based applications store CSS in a server-side database. Developers can update these CSS values and the application architecture can use server-side scripting to write the CSS values from the database to the desired webpage.

EditLive! provides the [setStyles Method](#) to allow developers to load CSS values directly into EditLive!. The CSS string passed to this property could be printed to the page using server-side scripting, as mentioned above. This would ensure the CSS applied to EditLive! would always be the most up to date CSS extracted from the server.

For this tutorial, no server side scripting is used. However, the CSS string

```
h5.warning{color:red;font-weight:bold;}
```

could easily have been written by a server-side script.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/styles_config.
xml");
      // Specifying CSS for use by editor
      editlivejava.setStyles(encodeURIComponent("h5.warning{color:red;font-weight:bold;}"));
      editlivejava.show();
    </script>
  </body>
</html>
```

# Setting CSS in the Applet Code

```
<!--
*****

setStyles.html --

This tutorial shows developers how to specify the CSS to be used by EditLive!.

Copyright © 2001-2012 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Setting the CSS to be Used by EditLive! - Tutorial</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Setting the CSS Used by EditLive!</h1>

    <p>This tutorial shows how to specify the CSS used by an instance of EditLive!</p>

    <p>The instance of EditLive! featured in this tutorial uses styles specified in a CSS file, as
    well styles explicitly passed to editor using it's load-time properties.</p>

    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
      a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationFile("styles_config.xml");

      // Specifying CSS for use by editor
      editlive.setStyles(encodeURIComponent("h5.warning{color:red;font-weight:bold;}"));

      // Before sending HTML to the instance of EditLive!, this HTML must be URL Encoded.
      // Javascript provides several URL Encoding methods, the best of which is
      // 'encodeURIComponent'
      editlive.setBody(encodeURIComponent("<h1>Specifying CSS for Use with EditLive!<
      /h1><h3>Using the EditLive! Configuration File</h3><p>The EditLive! Configuration File can be used to
      explicitly define CSS or to reference an external CSS file.</p><h3>Using the Styles Load-Time Property<
      /h3><p>The <em>Styles</em> load-time property for EditLive! can be used to specify CSS.</p><h5 class=\"warning\"
      >Note: The Styles load-time property cannot be used to reference an external CSS file.</h5>"));

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
      insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>
```

# Setting CSS in the Swing SDK

EditLive! for Java Swing allows developers to specify the CSS for rendering the HTML content created within the editor. For a complete list of the methods available for specifying CSS for use with EditLive! for Java Swing, see the [Using CSS in the Swing SDK](#) article in the [Developer Guide](#) section of this SDK.

This tutorial provides developers with the knowledge required to load CSS into EditLive! for Java Swing under the following circumstances:

- When CSS is stored in an external .css file, and
- When CSS is dynamically generated into the HTML page from a server-side script.

## Tutorial

The [Setting CSS in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to load CSS into EditLive! from either an external CSS file or as a defined string of CSS.

## Code

The complete code view for all the associated files in the [Setting CSS in the Swing SDK Tutorial](#) is available [here](#).



# Setting CSS in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Create a Reference to the External CSS File in the EditLive! Swing SDK Configuration File

To specify an external CSS file for use with the EditLive! Swing SDK, you'll need to make changes to your EditLive! Swing SDK Configuration File. As outlined in the [Creating and Editing Configuration Files Tutorial](#), this is done using a text editor.

Open the *sampleeljconfig.xml* file packaged with EditLive! using a text editor. Locate the following line of code:

```
<!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css" /> -->
```

Remove the <!-- and --> characters wrapping to tag.

```
<link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css" />
```

Change the **href** attribute to reference the *main.css* file stored on the Tiny website.

```
<link rel="stylesheet" href="http://www.ephox.com/images/main.css" type="text/css" />
```

Save this configuration file as *styles\_config.xml*.

### Step 2. Create an Instance of EditLive! for Java in a JFrame

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! for Java in a [JFrame](#).

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetStyles {
    /** html content to appear in the instance of EditLive! for Java */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! Swing SDK*/
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("styles_config.xml"), false);

    public SetStyles() throws Exception {
        super("Tutorial - Set Styles");
        this.getContentPane().setLayout(new FlowLayout());
    }
}
```

```

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetStyles();
    }
}

```

Ensure *styles\_config.xml* is specified as the EditLive! Swing SDK Configuration File.

### Step 3. Specify a String of CSS Using the `setStyles()` Method

EditLive! Swing SDK features a `setStyles()` method that allows developers to specify a string of CSS to be used for the rendering of the editor's HTML content. `setStyles()` allows developers to generate CSS from any location (for example, extracting CSS information stored in a database) and load this CSS into the EditLive! Swing SDK, rather than statically defining the CSS in an EditLive! for Java Configuration File.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class SetStyles {
    /** html content to appear in the instance of EditLive! Swing SDK */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
        "styles_config.xml"), false);

    public SetStyles() throws Exception {
        super("Tutorial - Set Styles");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {

```

```

        public void run() {
            // set CSS for EditLive! and initialize
            editLiveBean.setStyles("h5.warning{color:red;font-weight:bold;}");
            editLiveBean.init();
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new SetStyles();
}
}

```

# Setting CSS in the Swing SDK Code

```
/*
 * Copyright (c) 2006 Ephox Corporation.
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.epfox.editlive.*;
import javax.jnlp.*;

/** Class loads a JFrame with a single panel, containing the
 * Ephox EditLive! Editor
 */
public class SetStyles extends JFrame{
    /** html content to appear in the instance of EditLive! */
    private static final String INITIAL_HTML = "<h1>Specifying CSS for Use with EditLive!</h1><h3>Using the
EditLive! Configuration File</h3><p>The EditLive! Configuration File can be used to explicitly define CSS or to
reference an external CSS file.</p><h3>Using the Styles Load-Time Property</h3><p>The <em>Styles</em> load-time
property for EditLive! can be used to specify CSS.</p><h5 class=\"warning\">Note: The Styles load-time property
cannot be used to reference an external CSS file.</h5>";

    /** Base class for EditLive! */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("styles_config.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
    *
    */
    public SetStyles() throws Exception {
        super("Tutorial - Set Styles");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // set CSS for EditLive! and initialize
                editLiveBean.setStyles("h5.warning{color:red;font-weight:bold;}");
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins it's execution
    *
    * @param args the command line arguments - ignored
    */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
    }
}
```

```
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new SetStyles();
    }
}
```

# Custom Toolbar Buttons

- [Custom Toolbar Buttons in the Applet](#)
  - [Custom Toolbar Buttons in the Applet Tutorial](#)
  - [Custom Toolbar Buttons in the Applet Code](#)
    - [customItem.xml](#)
    - [customToolbarItem.html](#)
    - [mockDialog.html](#)
- [Custom Toolbar Buttons in the Swing SDK](#)
  - [Custom Toolbar Button in the Swing SDK Tutorial](#)
  - [Custom Toolbar Button in the Swing SDK Code](#)
    - [customItem.xml \(Java Swing\)](#)
    - [customToolbarItem.java](#)

# Custom Toolbar Buttons in the Applet

EditLive! allows developers to create their own custom toolbar buttons and menu items. These custom items can perform a number of different functions. For a complete list of the functionality available through custom toolbar buttons and menu items, see the [Creating Custom Menu and Toolbar Items](#) article in the [Developer Guide](#) section of this SDK.

This tutorial shows how to create a custom toolbar button that can call a Javascript function.

## Tutorial

The [Custom Toolbar Buttons in the Applet Tutorial](#) provides a step-by-step walk-through on how to specify both character encoding types for EditLive!.

## Code

The complete code views for all the associated files in the [Custom Toolbar Buttons in the Applet Tutorial](#) are available [here](#).

# Custom Toolbar Buttons in the Applet Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Creating a customToolBarButton in the EditLive! Configuration File

Open the *sampleeljconfig.xml* file packaged with EditLive! using a text editor. Locate the following line of code:

```
<toolbarButton name="Color" />
```

Add the following elements after the **Color** toolbar button:

```
<toolbarSeparator/>
<customToolBarButton
  name="displayDialog"
  text="Display Example Dialog"
  imageURL="images/small_logo.gif"
  action="raiseEvent"
  value="displayDialog"
/>
```

The above custom toolbar button defines the following:

- The unique name for the toolbar button is **displayDialog**.
- The tool-tip text that will appear when the mouse hovers over the button will be **Display Example Dialog**.
- The location of the image used in the custom toolbar button is *images/small\_logo.gif* (this location is relative to the webpage where EditLive! is being instantiated).
- This toolbar button will call to the Javascript method *displayDialog()*.

Save the file as *customItem.xml*.

### Step 2. Create an Instance of EditLive! in a Webpage

As shown in the [Instantiation Tutorial](#), create an instance of EditLive! in a webpage.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_customItem.xml");
      editlivejava.show();
    </script>
  </body>
</html>
```

Note that the *customItem.xml* file created above is the specified EditLive! Configuration File.



Save this webpage as *customToolBarItem.html*.

### Step 3. Create the displayDialog Javascript Method

As depicted in Step 1, the custom toolbar button will call the Javascript method *displayDialog()*. This method will display a new HTML page (*mockDialog.html*) in a new window.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_customItem.
xml");

      editlivejava.show();

      function displayDialog() {
        window.open('mockDialog.html','MockDialog','width=' + 300 + ',height=' + 175 + ',
status=no,resizable=yes,scrollbars=no,location=no,toolbar=no');
      }
    </script>
  </body>
</html>
```

### Step 4. Create a Javascript Method to be Called from the New Window

In order for the *mockDialog.html* page to depict a dialog, the new window will need to be able to call back to its parent page and perform an interaction with EditLive!.

The *insertString()* function created below will insert the string **dialog called** into the instance of EditLive!.

```
<html>
  <body>
    <script src="../../redistributables/editlivejava/editlivejava.js"
language="JavaScript"></script>
    <script>
      var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
      editlivejava.setConfigurationFile("../../redistributables/editlivejava/sample_customItem.
xml");

      editlivejava.show();

      function displayDialog() {
        window.open('mockDialog.html','MockDialog','width=' + 300 + ',height=' + 175 + ',
status=no,resizable=yes,scrollbars=no,location=no,toolbar=no');
      }

      function insertString() {
        editlive.insertHTMLAtCursor(encodeURIComponent("<b>dialog called</b>"));
      }
    </script>
  </body>
</html>
```

### Step 5. Create the mockDialog.html Webpage

The *mockDialog.html* page requires a control allowing the user to call back to the parent page and insert some text into EditLive!.

This can be performed through a simple Javascript routine triggered by a HTML button's **onClick** method.

```
<html>
  <head>
    <title>Example Dialog</title>
    <link rel="stylesheet" href="stylesheet.css">
  </head>
  <body>
    <h1>Example Dialog</h1>
```

```
<p>Click the OK button to load the string "&lt;b&gt;dialog called&lt;/b&gt;" into EditLive! and close this window.</p>
```

```
<input type="button" value="OK" onClick="window.opener.insertString();window.close()"/>
</body>
</html>
```

Save this page as *mockDialog.html*.

# Custom Toolbar Buttons in the Applet Code

- [customItem.xml](#)
- [customToolbarItem.html](#)
- [mockDialog.html](#)

# customItem.xml

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

<!--

This file customizes and configures EditLive!.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFFFCC;
          }
        </style>
        -->
      </head>

      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
    -->
  </document>
</editlive>
```

```

-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    accountID="BB56B8DD47EF"
    activationURL="http://www.ephox.com/elregister/el2/activate.asp"
    domain="LOCALHOST"
    expiration="NEVER"
    forceActive="false"
    key="6FFF-9765-8052-7AA5"
    licensee="For Evaluation Only"
    product="EditLive! for Java"
    release="6.0"
    seats=""
    type="Evaluation License"
    eqEditor="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
  tabPlacement="bottom"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false"
  enableTrackChanges="false"
>
  <!-- Define Custom Tags actions -->
  <!--
  <customTags>
    <doubleClickActions>
      <action../>
    </doubleClickActions>
  </customTags>
  -->

```

```

    <!-- Define additional symbols for the symbol dialog here -->
    <!--
    <symbols></symbols>
    -->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
    <!--
    Specify HTTP upload settings
    'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
    'href' is your server-side script for handling multipart-formdata uploads from ELJ
    -->

    <httpUpload
        base="http://www.yourserver.com/userfiles/"
        href="http://www.yourserver.com/scripts/upload.jsp">

        <!--
        Specify any additional fields to post with the image data
        -->
        <!--<httpUploadData name="hello" data="world"/> -->

    </httpUpload>

    <images allowLocalImages="true" allowUserSpecified="true">
        <!--
        The list of images which appear in the Insert Image dialog.
        TIP: Dynamically generate this from your database or repository to achieve an easy image library.
        -->

        <imageList>
            <image name="EditLive!"
                description="EditLive! Logo"
                alt="EditLive! Logo"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
                title="EditLive!" />

            <image name="iMac"
                alt="iMac Computer"
                description="iMac Computer"
                title="iMac"
                border="0"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

            <image name="Apple Computer"
                alt="Apple Computer"
                title="Apple Computer"
                description="Picture of a new Apple Computer"
                border="0"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

            <image name="IBM Thinkpad"
                alt="IBM Thinkpad"
                border="0"
                title="IBM Thinkpad"
                description="Picture of a new IBM Thinkpad"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"

```

```

        />

    </imageList>
</images>
<multimedia>
    <types>
        <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
            <param name="movie" />
            <param name="quality" />
            <param name="bgcolor" />
        </type>
        <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
            <param name="autohref" />
            <param name="autoplay" />
            <param name="bgcolor" />
            <param name="cache" />
            <param name="controller" />
            <param name="correction" />
            <param name="dontflattenwhensaving" />
            <param name="enablejavascript" />
            <param name="endtime" />
            <param name="fov" />
            <param name="height" />
            <param name="href" />
            <param name="kioskmode" />
            <param name="loop" />
            <param name="movieid" />
            <param name="moviename" />
            <param name="node" />
            <param name="pan" />
            <param name="playeveryframe" />
            <param name="qtsrccchokespeed" />
            <param name="scale" />
            <param name="starttime" />
            <param name="target" />
            <param name="targetcache" />
            <param name="tilt" />
            <param name="urlsubstitute" />
            <param name="volume" />
        </type>
        <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
            <param name="animationAtStart" />
            <param name="autoStart" />
            <param name="showControls" />
            <param name="clickToPlay" />
            <param name="transparentAtStart" />
        </type>
        <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
        <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
        <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
    </types>
</multimedia>
</mediaSettings>

<hyperlinks>

    <hyperlinkList>
        <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
        <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
        <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
    </hyperlinkList>

```

```

</hyperlinkList>

<mailtoList>
  <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
</mailtoList>

</hyperlinks>

<!--
Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

  <menu name="ephox_filemenu">
    <menuItem name="New" />
    <menuItem name="Open" />
    <menuSeparator />
    <menuItem name="Save" />
    <menuItem name="SaveAs" />
    <menuSeparator />
    <menuItem name="Print" />
  </menu>

  <menu name="ephox_editmenu">
    <menuItem name="Undo" />
    <menuItem name="Redo" />
    <menuSeparator />
    <menuItem name="Cut" />
    <menuItem name="Copy" />
    <menuItem name="Paste" />
    <menuItem name="PasteSpecial" />
    <menuSeparator />
    <menuItem name="Select" />
    <menuItem name="SelectAll" />
    <menuSeparator />
    <menuItem name="Find" />
    <menuSeparator />
  </menu>

  <menu name="ephox_viewmenu">
    <menuItemGroup name="SourceView" />
    <menuSeparator />
    <menuItem name="Popout" />
    <menuSeparator />
    <menuItem name="showDocumentNavigator" />
    <menuSeparator />
    <menuItem name="ParagraphMarker" />
  </menu>

  <menu name="ephox_insertmenu">

    <menuItem name="HLink" />
    <menuItem name="Bookmark" />
    <menuItem name="RemoveHyperlink" />
    <menuSeparator />
    <menuItem name="ImageServer" />
    <menuItem name="InsertObject" />
    <menuSeparator />
    <menuItem name="Symbol" />
    <menuItem name="HRule" />
    <menuSeparator />
    <menuItem name="DateTime" />
    <menuSeparator />
    <menuItem name="insertcomment" />
  </menu>

```



```

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>

  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>

  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuItem name="tableautofit"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>
  <menuItem name="InsTextField"/>
  <menuItem name="InsPasswordField"/>
  <menuItem name="InsHiddenField"/>
  <menuItem name="InsFileField"/>
  <menuItem name="InsButtonField"/>
  <menuItem name="InsSubmitField"/>
  <menuItem name="InsResetField"/>
  <menuItem name="InsCheckboxField"/>
  <menuItem name="InsRadioField"/>
  <menuItem name="InsTextAreaField"/>
  <menuItem name="InsSelectField"/>
  <menuItem name="InsImageField"/>
</menu>

<menu name="ephox_trackchangesmenu">

```

```

        <menuItem name="enabletrackchanges" />
        <menuSeparator />
        <menuItem name="acceptChange" />
        <menuItem name="rejectChange" />
        <menuSeparator />
        <menuItem name="previousChange" />
        <menuItem name="nextChange" />
        <menuSeparator />
        <menuItem name="acceptAllChanges" />
        <menuItem name="rejectAllChanges" />
        <menuSeparator />
        <menuItem name="showTrackChangesDialog" />
        <menuSeparator />
        <menuItem name="setUsername" />
    </menu>
</menuBar>

```

```

<!--

```

```

Customize the EditLive! toolbars
-->

```

```

-->

```

```

<toolbars>

```

```

    <toolbar name="Command">

```

```

        <toolbarButton name="Print" />
        <toolbarSeparator />
        <toolbarButton name="Spelling" />

```

```

        <toolbarButton name="Find" />
        <toolbarSeparator />
        <toolbarButton name="Cut" />
        <toolbarButton name="Copy" />
        <toolbarButton name="Paste" />
        <toolbarButton name="FormatPainter" />
        <toolbarSeparator />
        <toolbarButton name="Undo" />
        <toolbarButton name="Redo" />
        <toolbarSeparator />
        <toolbarButton name="HLink" />
        <toolbarButton name="ImageServer" />
        <toolbarButton name="insertequation" />
        <toolbarSeparator />
        <toolbarButton name="InsTableWizard" />
        <toolbarButton name="InsRow" />
        <toolbarButton name="InsCol" />
        <toolbarButton name="DelRow" />
        <toolbarButton name="DelCol" />
        <toolbarSeparator />

```

```

        <toolbarButton name="enableTrackChanges" />
        <toolbarButton name="acceptChange" />
        <toolbarButton name="rejectchange" />
        <toolbarButton name="previouschange" />
        <toolbarButton name="nextchange" />
        <toolbarSeparator />
        <toolbarButton name="ParagraphMarker" />
        <toolbarSeparator />

```

```

        <toolbarButton name="Popout" />
    </toolbar>

```

```

</toolbar>

```

```

<toolbar name="Format">

```

```

    <!--

```

```

        Styles from any embedded or external stylesheets will also be automatically added to the Styles

```

drop-down

```

-->

```

```

        <toolbarComboBox name="Style">
            <comboBoxItem name="P" />
            <comboBoxItem name="H1" />

```

```

        <comboBoxItem name="H2" />
        <comboBoxItem name="H3" />
        <comboBoxItem name="H4" />
        <comboBoxItem name="H5" />
        <comboBoxItem name="H6" />
    </toolbarComboBox>
    <!--
    You can remove the Font drop-down if you just want users to use Styles.
    The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
and Windows
    To change the default font, change the embedded style sheet in the 'style' element above.
    -->
    <toolbarComboBox name="Face">
        <comboBoxItem name="Arial" text="Arial" />
        <comboBoxItem name="Arial Black" text="Arial Black" />
        <comboBoxItem name="Arial Narrow" text="Arial Narrow" />
        <comboBoxItem name="Comic Sans MS" text="Comic Sans MS" />
        <comboBoxItem name="Courier New" text="Courier New" />
        <comboBoxItem name="Georgia" text="Georgia" />
        <comboBoxItem name="Impact" text="Impact" />
        <comboBoxItem name="Times New Roman" text="Times New Roman" />
        <comboBoxItem name="Trebuchet MS" text="Trebuchet MS" />
        <comboBoxItem name="Verdana" text="Verdana" />
    </toolbarComboBox>
    <!--
    Font Size drop-down
    -->
    <toolbarComboBox name="Size">
        <comboBoxItem name="1" text="8pt" />
        <comboBoxItem name="2" text="10pt" />
        <comboBoxItem name="3" text="12pt" />
        <comboBoxItem name="4" text="14pt" />
        <comboBoxItem name="5" text="18pt" />
        <comboBoxItem name="6" text="24pt" />
        <comboBoxItem name="7" text="36pt" />
    </toolbarComboBox>
    <toolbarSeparator />
    <toolbarButton name="Bold" />
    <toolbarButton name="Italic" />
    <toolbarButton name="Underline" />
    <toolbarSeparator />
    <toolbarButtonGroup name="Align" />
    <toolbarSeparator />
    <toolbarButtonGroup name="List" />
    <toolbarButton name="DecreaseIndent" />
    <toolbarButton name="IncreaseIndent" />
    <toolbarSeparator />
    <toolbarButton name="HighlightColor" />
    <toolbarButton name="Color" />
    <toolbarSeparator />
    <customToolbarButton
        name="displayDialog"
        text="Display Example Dialog"
        imageURL="images/small_logo.gif"
        action="raiseEvent"
        value="displayDialog"
    />
</toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
    <shrtMenu>
        <shrtMenuItem name="Undo" />
        <shrtMenuItem name="Redo" />
        <shrtMenuSeparator />
        <shrtMenuItem name="Cut" />
        <shrtMenuItem name="Copy" />
        <shrtMenuItem name="Paste" />
    </shrtMenu>
</shortcutMenu>

```

```
<shrtMenuSeparator/>
<shrtMenuItem name="Select" />
<shrtMenuSeparator/>
<shrtMenuItem name="acceptChange" />
<shrtMenuItem name="rejectChange" />
<shrtMenuItem name="nextchange" />
<shrtMenuItem name="previouschange" />
<shrtMenuSeparator/>
<shrtMenuItem name="Hyperlink" />
<shrtMenuItem name="RemoveHyperlink" />
<shrtMenuItem name="PropImage" />
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator/>
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator/>
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator/>
    <shrtMenuItem name="synonyms" />
<shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# customToolbarItem.html

```
<!--
*****

customToolbarItem.html --

EditLive! tutorial to use only the most basic
javascript methods to instantiate the editor in a webpage

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Custom Toolbar Buttons</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Creating Custom Toolbar Buttons</h1>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
      a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationFile("customItem.xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
      insert a new EditLive! instance
      // at the this location.
      editlive.show();

      /** This function will open the mockDialog.html page in a new window.
      *
      */
      function displayDialog() {
        window.open('mockDialog.html','MockDialog','width=' + 300 + ',height=' + 175 + ',
status=no,resizable=yes,scrollbars=no,location=no,toolbar=no');
      }

      /** This function is called to from the mockDialog.html window.
      *
      */
      function insertString() {
        editlive.insertHTMLAtCursor(encodeURIComponent("<b>dialog called</b>"));
      }
    </script>
  </body>
</html>
```

# mockDialog.html

```
<html>
  <head>
    <title>Example Dialog</title>
    <link rel="stylesheet" href="stylesheet.css">
  </head>
  <body>
    <h1>Example Dialog</h1>

    <p>Click the OK button to load the string "&lt;b&gt;dialog called&lt;/b&gt;" into EditLive! and
close this window.</p>

    <input type="button" value="OK" onClick="window.opener.insertString();window.close()"/>
  </body>
</html>
```

# Custom Toolbar Buttons in the Swing SDK

EditLive! for Java Swing allows developers to create their own custom toolbar buttons and menu items. These custom items can perform a number of different functions. For a complete list of the functionality available through custom toolbar buttons and menu items, see the [Creating Custom Menu and Toolbar Items](#) article in the [Developer Guide](#) section of this SDK.

This tutorial shows how to create a custom toolbar button that can display a custom created dialog.

## Tutorial

The [Custom Toolbar Buttons in the Swing SDK Tutorial](#) provides a step-by-step walk-through on how to specify both character encoding types for EditLive!.

## Code

The complete code views for all the associated files in the [Custom Toolbar Buttons in the Swing SDK Tutorial](#) are available [here](#).

# Custom Toolbar Button in the Swing SDK Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic Java programming with the Swing library
- Basic Knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Creating and Editing Configuration Files Tutorial](#)

## Tutorial

### Step 1. Creating a customToolbarButton in the EditLive! Configuration File

Open the *sampleljconfig.xml* file packaged with EditLive! for Java Swing using a text editor. Locate the following line of code:

```
<toolbarButton name="Color" />
```

Add the following elements after the **Color** toolbar button:

```
<toolbarSeparator/>
<customToolbarButton
  name="displayDialog"
  text="Display Example Dialog"
  imageURL="http://www.ephox.com/images/small_logo.gif"
  action="raiseEvent"
  value="displayDialog"
/>
```

The above custom toolbar button defines the following:

- The unique name for the toolbar button is **displayDialog**.
- The tool-tip text that will appear when the mouse hovers over the button will be **Display Example Dialog**.
- The location of the image used in the custom toolbar button is *images/small\_logo.gif* (this location is relative to the webpage where EditLive! is being instantiated).
- This toolbar button will raise a [TextEvent](#) with the following parameters:
  - Action Command: **TextEvent.CUSTOM\_ACTION**
  - Extra String: **displayDialog**.

Save the file as *customItem.xml*.

### Step 2. Create an Instance of EditLive! for Java Swing in a JFrame

As shown in the Instantiation Tutorial, create an instance of EditLive! for Java Swing in a [JFrame](#).

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CustomToolbarItem {
  /** html content to appear in the instance of EditLive! for Java Swing */
  private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";
```



```

    /** Base class for EditLive! for Java */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("customItem.xml"), false);

    public CustomToolBarItem() throws Exception {
        super("Tutorial - Custom Toolbar Items");
        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                editLiveBean.init();
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** Sets up the application and begins its execution
    *
    * @param args the command line arguments - ignored
    */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new CustomToolBarItem();
    }
}

```

Note that the *customItem.xml* EditLive! for Java Swing Configuration File created above is specified.

Save this class as *CustomToolBarItem.java*.

### Step 3. Implement an EventListener Interface and Create the raiseEvent Method

As mentioned in step 1, the custom toolbar button will create a *TextEvent* and send this event to all registered listeners. *TextEvent* are listened to by classes extending the [EventListener](#) interface. The *raiseEvent* method of the *EventListener* interface is responsible for handling these *TextEvents*.

For the *CustomToolBarItem* class, we want to catch only *TextEvents* with the parameters matching those specified for our custom toolbar button.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CustomToolBarItem implements EventListener {
    /** html content to appear in the instance of EditLive! for Java Swing */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";
}

```

```

/** Base class for EditLive! for Java Swing */
private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("customItem.xml"), false);

public CustomToolBarItem() throws Exception {
    super("Tutorial - Custom Toolbar Items");

    this.getContentPane().setLayout(new FlowLayout());

    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            // initialize EditLive! and add this class as an event listener
            editLiveBean.init();
            editLiveBean.addEditorEventListener(CustomToolBarItem.
this);
        }
    });

    // Create a JPanel to hold the ELJBean
    JPanel editorHolder = new JPanel(new FlowLayout());
    editorHolder.add(editLiveBean);
    // Add the JPanel to the JFrame
    this.getContentPane().add(editorHolder);

    // Display the JFrame.
    this.setSize(new Dimension(710, 620));
    this.setVisible(true);

    // Adding a listener to detect if the JFrame is closing, to close the application if needed.
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

/** raiseEvent method inherited from EventListener. Every action in EditLive! for Java Swing
 * raises a TextEvent.
 *
 * @param e TextEvent information sent on action.
 */
public void raiseEvent(TextEvent e) {
    if(e.getActionCommand() == TextEvent.CUSTOM_ACTION && e.getExtraString().equals
("displayDialog")) {

    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new CustomToolBarItem();
}
}

```

#### Step 4. Create a New JFrame to Display

When a user clicks the custom toolbar button, the desired result will be to display a new dialog. To do this, a new class will need to be created that displays a JFrame.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CustomToolBarItem implements ActionListener {
    /** html content to appear in the instance of EditLive! for Java Swing */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java Swing */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("customItem.xml"), false);

    public CustomToolBarItem() throws Exception {
        super("Tutorial - Custom Toolbar Items");

        editLiveBean.addEditorEventListener(this);

        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive! and add this class as an event listener
                editLiveBean.init();
                editLiveBean.addEditorEventListener(CustomToolBarItem.this);
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** raiseEvent method inherited from ActionListener. Every action in EditLive! for Java Swing
     * raises a TextEvent.
     *
     * @param e TextEvent information sent on action.
     */
    public void raiseEvent(TextEvent e) {
        if(e.getActionCommand() == TextEvent.CUSTOM_ACTION && e.getExtraString().equals("displayDialog")) {

        }
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {

```

```

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }
    new CustomToolBarItem();
}

class ExampleDialog extends JFrame {
    private JLabel headingLabel = new JLabel("Example Dialog");
    private JLabel infoLabel = new JLabel("Click the OK button to load the string \"<b>dialog called</b>\"
into EditLive! and close this window.");
    private JButton OKButton = new JButton("OK");

    ExampleDialog() {
        super("Example Dialog");

        headingLabel.setFont(new java.awt.Font("Arial", Font.BOLD, 14));

        this.getContentPane().setLayout(new GridLayout(3, 1));
        this.getContentPane().add(headingLabel);
        this.getContentPane().add(infoLabel);

        JPanel buttonPanel = new JPanel(new FlowLayout());
        buttonPanel.add(OKButton);

        this.getContentPane().add(buttonPanel);

        // Display the JFrame.
        this.setSize(new Dimension(500, 175));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
    }
}

```

## Step 5. Call the New JFrame

The *ExampleDialog* class can be called from the **raiseEvent** method. In order to ensure the *ExampleDialog* class can communicate with the instance of the EditLive! for Java JavaBean.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CustomToolBarItem implements EventListener {
    /** html content to appear in the instance of EditLive! for Java Swing */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java Swing */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource(
("customItem.xml"), false);

    public CustomToolBarItem() throws Exception {
        super("Tutorial - Custom Toolbar Items");

        editLiveBean.addEditorEventListener(this);
    }
}

```

```

        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive! and add this class as an event listener
                editLiveBean.init();
                editLiveBean.addEditorEventListener(CustomToolBarItem.
this);
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** raiseEvent method inherited from EventListener. Every action in EditLive! for Java
    * raises a TextEvent.
    *
    * @param e TextEvent information sent on action.
    */
    public void raiseEvent(TextEvent e) {
        if(e.getActionCommand() == TextEvent.CUSTOM_ACTION && e.getExtraString().equals
("displayDialog")) {
            ExampleDialog exDialog = new ExampleDialog(editLiveBean);
            exDialog.setVisible(true);
        }
    }

    /** Sets up the application and begins its execution
    *
    * @param args the command line arguments - ignored
    */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new CustomToolBarItem();
    }
}

class ExampleDialog extends JFrame {
    // Base class for EditLive! for Java Swing
    private ELJBean elBean;

    private JLabel headingLabel = new JLabel("Example Dialog");
    private JLabel infoLabel = new JLabel("Click the OK button to load the string \<b>dialog called</b>\"
into EditLive! and close this window.");
    private JButton OKButton = new JButton("OK");

    ExampleDialog() {
        super("Example Dialog");
        elBean = _elBean;
    }
}

```

```

        headingLabel.setFont(new java.awt.Font("Arial", Font.BOLD, 14));

        this.getContentPane().setLayout(new GridLayout(3, 1));
        this.getContentPane().add(headingLabel);
        this.getContentPane().add(infoLabel);

        JPanel buttonPanel = new JPanel(new FlowLayout());
        buttonPanel.add(OKButton);

        this.getContentPane().add(buttonPanel);

        // Display the JFrame.
        this.setSize(new Dimension(500, 175));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
    }
}

```

## Step 6. Extend the ActionListener Interface for the ExampleDialog Class

The final step of this tutorial is to extend the ActionListener interface for the *ExampleDialog* class. This will allow actions fired from the JButton in the dialog to then interact with the instance of EditLive! for Java Swing in the parent JFrame. The **actionPerformed** method provided by the ActionListener will use the insertHTMLAtCursor method of the EditLive! for Java JavaBean.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.ephox.editlive.*;

public class CustomToolBarItem implements ActionListener {
    /** html content to appear in the instance of EditLive! for Java Swing */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java Swing */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource("customItem.xml"), false);

    public CustomToolBarItem() throws Exception {
        super("Tutorial - Custom Toolbar Items");

        editLiveBean.addEditorEventListener(this);

        this.getContentPane().setLayout(new FlowLayout());

        SwingUtilities.invokeLaterAndWait(new Runnable() {
            public void run() {
                // initialize EditLive! and add this class as an event listener
                editLiveBean.init();
                editLiveBean.addEditorEventListener(CustomToolBarItem.this);
            }
        });

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);
    }
}

```

```

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** raiseEvent method inherited from ActionListener. Every action in EditLive! for Java
     * raises a TextEvent.
     *
     * @param e TextEvent information sent on action.
     */
    public void raiseEvent(TextEvent e) {
        if(e.getActionCommand() == TextEvent.CUSTOM_ACTION && e.getExtraString().equals
("displayDialog")) {
            ExampleDialog exDialog = new ExampleDialog(editLiveBean);
            exDialog.setVisible(true);
        }
    }

    /** Sets up the application and begins its execution
     *
     * @param args the command line arguments - ignored
     */
    public static void main(String args[]) throws Exception {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch(Exception e) {
            System.out.println("Unable to locate UIManager class");
            e.printStackTrace();
        }
        new CustomToolBarItem();
    }
}

class ExampleDialog extends JFrame implements ActionListener {
    // Base class for EditLive! for Java Swing
    private ELJBean elBean;

    private JLabel headingLabel = new JLabel("Example Dialog");
    private JLabel infoLabel = new JLabel("Click the OK button to load the string \"<b>dialog called</b>\"
into EditLive! and close this window.");
    private JButton OKButton = new JButton("OK");

    ExampleDialog() {
        super("Example Dialog");
        elBean = _elBean;

        OKButton.addActionListener(this);
        headingLabel.setFont(new java.awt.Font("Arial", Font.BOLD, 14));

        this.getContentPane().setLayout(new GridLayout(3, 1));
        this.getContentPane().add(headingLabel);
        this.getContentPane().add(infoLabel);

        JPanel buttonPanel = new JPanel(new FlowLayout());
        buttonPanel.add(OKButton);

        this.getContentPane().add(buttonPanel);

        // Display the JFrame.
        this.setSize(new Dimension(500, 175));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.

```

```

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
     *
     * @param e ActionEvent sent by JButton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == OKButton) {
            try {
                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        elBean.insertHTMLAtCursor("<b>dialog called</b>");

                        // close window
                        dispose();
                    }
                });
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }
    }
}

```



# Custom Toolbar Button in the Swing SDK Code

- [customItem.xml \(Java Swing\)](#)
- [customToolbarItem.java](#)

# customItem.xml (Java Swing)

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

<!--

This file customizes and configures EditLive! for Java Swing.

TIP: this file can be dynamically generated using ASP, JSP or PHP to achieve runtime changes to settings

-->
<editlive>

  <!-- Default content for the editor -->
  <document>
    <html>

      <!--
      Default document header
      -->
      <head>

        <!--
        Specify the base URL for the editor to download all relative images and style sheets
        -->
        <!--<base href="http://www.yourserver.com/cms/" />-->

        <!--
        Specify the character encoding for the editor. By default this should be UTF-8, which
        will encode all special characters as numeric entities in XHTML or as named entities in HTML
        -->
        <!--<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />-->

        <!--
        Uncomment the following line to load an external style sheet for the editor
        -->
        <!-- <link rel="stylesheet" href="http://www.yourserver.com/style.css" type="text/css"/> -->

        <!--
        Specify any embedded styles for the editor
        You can remove or customize the styles below.
        -->
        <!--
        <style type="text/css">
          body {
            font-family: Verdana, Arial;
          }
          h1 {
            font-family: Tahoma, Arial;
            font-size: 24pt;
            font-weight: normal;
            color: #003366;
            border-bottom: solid 1px #003366;
          }
          p.fineprint{
            font-size: 8pt;
            text-align: center;
          }
          span.comment {
            border: solid 1px #FFFF00;
            background-color: #FFFFCC;
          }
        </style>
        -->
      </head>

      <!--
      Default document body. Add content here if you want this to be the default when the editor
      loads, although this is better done at runtime.
    -->
  </document>
</editlive>
```

```

-->
<body>
</body>

</html>
</document>

<!--
Add your Ephox-provided license key here
-->
<ephoxLicenses>
  <license
    accountID="BB56B8DD47EF"
    activationURL="http://www.ephox.com/elregister/el2/activate.asp"
    domain="LOCALHOST"
    expiration="NEVER"
    forceActive="false"
    key="6FFF-9765-8052-7AA5"
    licensee="For Evaluation Only"
    product="EditLive! for Java Swing"
    release="6.0"
    seats=""
    type="Evaluation License"
    eqEditor="true"
  />
</ephoxLicenses>

<!--
Specify the location of the spell checker and thesaurus.
If no spellcheck or thesaurus jars are specified, the location for these jars is automatically generated
based on the specified DownloadDirectory load-time property and the user's locale.
-->
<!--
<spellCheck jar="../../redistributables/editlivejava/dictionaries/en_us_4_0.jar" useNotModified="false">
</spellCheck>
<thesaurus jar="../../redistributables/editlivejava/thesaurus/thes_am_6_0.jar" useNotModified="false"/>
-->

<!--
Specify HTML filter settings
-->
<htmlFilter
  outputXHTML="true"
  outputXML="false"
  indentContent="false"
  logicalEmphasis="true"
  quoteMarks="false"
  uppercaseTags="false"
  uppercaseAttributes="false"
  wrapLength="0">
</htmlFilter>

<!--
Specify settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="off" to disable the tabs.
-->
<wysiwygEditor
  tabPlacement="bottom"
  brOnEnter="false"
  showDocumentNavigator="false"
  disableInlineImageResizing="false"
  disableInlineTableResizing="false"
  enableTrackChanges="false"
>
  <!-- Define Custom Tags actions -->
  <!--
  <customTags>
    <doubleClickActions>
      <action../>
    </doubleClickActions>
  </customTags>
  -->

```

```

    <!-- Define additional symbols for the symbol dialog here -->
    <!--
    <symbols></symbols>
    -->
</wysiwygEditor>
<!--
Specify settings for the Source (code) view of the editor
-->
<sourceEditor showBodyOnly="false"/>

<!--
Specify options for content that EditLive has detected has been pasted from Microsoft Word
-->
<wordImport styleOption="merge_inline_styles"/>

<!--
Specify options for content that EditLive has detected has been pasted from another HTML document
-->
<htmlImport styleOption="merge_inline_styles"/>

<mediaSettings>
    <!--
    Specify HTTP upload settings
    'base' is the base URL used to update the 'src' attributes of any local files in the HTML source
    'href' is your server-side script for handling multipart-formdata uploads from ELJ
    -->

    <httpUpload
        base="http://www.yourserver.com/userfiles/"
        href="http://www.yourserver.com/scripts/upload.jsp">

        <!--
        Specify any additional fields to post with the image data
        -->
        <!--<httpUploadData name="hello" data="world"/> -->

    </httpUpload>

    <images allowLocalImages="true" allowUserSpecified="true">
        <!--
        The list of images which appear in the Insert Image dialog.
        TIP: Dynamically generate this from your database or repository to achieve an easy image library.
        -->

        <imageList>
            <image name="Ephox EditLive! for Java Swing"
                description="Ephox EditLive! for Java Swing Logo"
                alt="Ephox EditLive! for Java Swing Logo"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/eljlogo.jpg"
                title="Ephox EditLive! for Java Swing" />

            <image name="iMac"
                alt="iMac Computer"
                description="iMac Computer"
                title="iMac"
                border="0"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/newimac.gif" />

            <image name="Apple Computer"
                alt="Apple Computer"
                title="Apple Computer"
                description="Picture of a new Apple Computer"
                border="0"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg" />

            <image name="IBM Thinkpad"
                alt="IBM Thinkpad"
                border="0"
                title="IBM Thinkpad"
                description="Picture of a new IBM Thinkpad"
                src="http://www.ephox.com/product/editliveforjava/demos/startcms/images/ibm_thinkpad.gif"

```

```

        />

    </imageList>
</images>
<multimedia>
  <types>
    <type name="Macromedia Flash" type="application/x-shockwave-flash" extension="swf"
allowCustomParams="true" urlParam="movie">
      <param name="movie" />
      <param name="quality" />
      <param name="bgcolor" />
    </type>
    <type name="QuickTime Movie" type="video/quicktime" extension="mov" allowCustomParams="true">
      <param name="autohref" />
      <param name="autoplay" />
      <param name="bgcolor" />
      <param name="cache" />
      <param name="controller" />
      <param name="correction" />
      <param name="dontflattenwhensaving" />
      <param name="enablejavascript" />
      <param name="endtime" />
      <param name="fov" />
      <param name="height" />
      <param name="href" />
      <param name="kioskmode" />
      <param name="loop" />
      <param name="movieid" />
      <param name="moviename" />
      <param name="node" />
      <param name="pan" />
      <param name="playeveryframe" />
      <param name="qtsrccchokespeed" />
      <param name="scale" />
      <param name="starttime" />
      <param name="target" />
      <param name="targetcache" />
      <param name="tilt" />
      <param name="urlsubstitute" />
      <param name="volume" />
    </type>
    <type name="Window Media" type="application/x-mplayer2" extension="asf" allowCustomParams="
true" urlParam="fileName">
      <param name="animationAtStart" />
      <param name="autoStart" />
      <param name="showControls" />
      <param name="clickToPlay" />
      <param name="transparentAtStart" />
    </type>
    <type name="Window Media (Streaming)" type="application/x-mplayer2" extension="asx"
allowCustomParams="true" urlParam="fileName">
      <param name="animationAtStart" />
      <param name="autoStart" />
      <param name="showControls" />
      <param name="clickToPlay" />
      <param name="transparentAtStart" />
    </type>
    <type name="WAV Audio" type="application/x-mplayer2" extension="wav" allowCustomParams="true" />
    <type name="MP3 Audio" type="application/x-mplayer2" extension="mp3" allowCustomParams="true" />
    <type name="AVI" type="application/x-mplayer2" extension="avi" allowCustomParams="true" />
  </types>
</multimedia>
</mediaSettings>

<hyperlinks>

  <hyperlinkList>
    <hyperlink href="http://www.ephox.com" description="Ephox Web site" />
    <hyperlink href="http://www.apple.com" description="Apple Computer Web site" />
    <hyperlink href="http://www.sun.com" description="Sun Microsystems Web site" />
  </hyperlinkList>

```

```

</hyperlinkList>

<mailtoList>
  <mailtoLink href="mailto:info@ephox.com" description="Ephox information" />
</mailtoList>

</hyperlinks>

<!--
Customize the EditLive! menus

Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="false") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application.
-->
<menuBar showAboutMenu="true">

  <menu name="ephox_filemenu">
    <menuItem name="New" />
    <menuItem name="Open" />
    <menuSeparator />
    <menuItem name="Save" />
    <menuItem name="SaveAs" />
    <menuSeparator />
    <menuItem name="Print" />
  </menu>

  <menu name="ephox_editmenu">
    <menuItem name="Undo" />
    <menuItem name="Redo" />
    <menuSeparator />
    <menuItem name="Cut" />
    <menuItem name="Copy" />
    <menuItem name="Paste" />
    <menuItem name="PasteSpecial" />
    <menuSeparator />
    <menuItem name="Select" />
    <menuItem name="SelectAll" />
    <menuSeparator />
    <menuItem name="Find" />
    <menuSeparator />
  </menu>

  <menu name="ephox_viewmenu">
    <menuItemGroup name="SourceView" />
    <menuSeparator />
    <menuItem name="Popout" />
    <menuSeparator />
    <menuItem name="showDocumentNavigator" />
    <menuSeparator />
    <menuItem name="ParagraphMarker" />
  </menu>

  <menu name="ephox_insertmenu">

    <menuItem name="HLink" />
    <menuItem name="Bookmark" />
    <menuItem name="RemoveHyperlink" />
    <menuSeparator />
    <menuItem name="ImageServer" />
    <menuItem name="InsertObject" />
    <menuSeparator />
    <menuItem name="Symbol" />
    <menuItem name="HRule" />
    <menuSeparator />
    <menuItem name="DateTime" />
    <menuSeparator />
    <menuItem name="insertcomment" />
  </menu>

```

```

<menu name="ephox_formatmenu">
  <submenu name="Style"/>
  <submenu name="Face"/>
  <submenu name="Size"/>
  <menuSeparator/>
  <menuItem name="Bold"/>
  <menuItem name="Italic"/>
  <menuItem name="Underline"/>
  <menuSeparator/>
  <menuItemGroup name="Align"/>
  <menuSeparator/>
  <menuItemGroup name="List"/>
  <menuItem name="DecreaseIndent"/>
  <menuItem name="IncreaseIndent"/>
  <menuItem name="PropList"/>
  <menuSeparator/>
  <menuItem name="HighlightColor"/>
  <menuItem name="Color"/>
  <menuSeparator/>
  <menuItemGroup name="Script"/>
  <menuItem name="Strike"/>
  <menuSeparator/>
  <menuItem name="RemoveFormatting"/>
  <menuItem name="FormatPainter"/>
</menu>

<menu name="ephox_toolsmenu">
  <menuItem name="Spelling"/>
  <menuItem name="BackgroundSpellChecking"/>
  <menuItem name="thesaurus"/>
  <menuSeparator/>
  <menuItem name="Accessibility"/>
  <menuSeparator/>
  <menuItem name="WordCount"/>
</menu>

<menu name="ephox_tablemenu">
  <menuItem name="InsTable"/>
  <menuItem name="InsRowCol"/>

  <menuSeparator/>
  <menuItem name="DelRow"/>
  <menuItem name="DelCol"/>

  <menuSeparator/>
  <menuItem name="Split"/>
  <menuItem name="Merge"/>
  <menuItem name="tableautofit"/>
  <menuSeparator/>
  <menuItem name="PropCell"/>
  <menuItem name="PropRow"/>
  <menuItem name="PropCol"/>
  <menuItem name="PropTable"/>
  <menuSeparator/>
  <menuItem name="Gridlines"/>
</menu>

<menu name="ephox_formmenu">
  <menuItem name="InsForm"/>
  <menuSeparator/>
  <menuItem name="InsTextField"/>
  <menuItem name="InsPasswordField"/>
  <menuItem name="InsHiddenField"/>
  <menuItem name="InsFileField"/>
  <menuItem name="InsButtonField"/>
  <menuItem name="InsSubmitField"/>
  <menuItem name="InsResetField"/>
  <menuItem name="InsCheckboxField"/>
  <menuItem name="InsRadioField"/>
  <menuItem name="InsTextAreaField"/>
  <menuItem name="InsSelectField"/>

```

```

        <menuItem name="InsImageField"/>
</menu>
<menu name="ephox_trackchangesmenu">
    <menuItem name="enabletrackchanges" />
    <menuSeparator />
    <menuItem name="acceptChange" />
    <menuItem name="rejectChange" />
    <menuSeparator />
    <menuItem name="previousChange" />
    <menuItem name="nextChange" />
    <menuSeparator />
    <menuItem name="acceptAllChanges" />
    <menuItem name="rejectAllChanges" />
    <menuSeparator />
    <menuItem name="showTrackChangesDialog" />
    <menuSeparator />
    <menuItem name="setUsername" />
</menu>
</menuBar>

<!--
Customize the EditLive! toolbars
-->
<toolbars>
    <toolbar name="Command">

        <toolbarButton name="Print"/>
        <toolbarSeparator/>
        <toolbarButton name="Spelling"/>

        <toolbarButton name="Find"/>
        <toolbarSeparator/>
        <toolbarButton name="Cut"/>
        <toolbarButton name="Copy"/>
        <toolbarButton name="Paste"/>
        <toolbarButton name="FormatPainter" />
        <toolbarSeparator/>
        <toolbarButton name="Undo"/>
        <toolbarButton name="Redo"/>
        <toolbarSeparator/>
        <toolbarButton name="HLink"/>
        <toolbarButton name="ImageServer"/>
        <toolbarButton name="insertequation"/>
        <toolbarSeparator/>
        <toolbarButton name="InsTableWizard"/>
        <toolbarButton name="InsRow"/>
        <toolbarButton name="InsCol"/>
        <toolbarButton name="DelRow"/>
        <toolbarButton name="DelCol"/>
        <toolbarSeparator/>
        <toolbarButton name="enableTrackChanges"/>
        <toolbarButton name="acceptChange"/>
        <toolbarButton name="rejectchange"/>
        <toolbarButton name="previouschange"/>
        <toolbarButton name="nextchange"/>
        <toolbarSeparator/>
        <toolbarButton name="ParagraphMarker"/>
        <toolbarSeparator/>

        <toolbarButton name="Popout"/>
    </toolbar>

    <toolbar name="Format">
        <!--
        Styles from any embedded or external stylesheets will also be automatically added to the Styles
drop-down
        -->

```



```

<toolbarComboBox name="Style">
  <comboBoxItem name="P" />
  <comboBoxItem name="H1" />
  <comboBoxItem name="H2" />
  <comboBoxItem name="H3" />
  <comboBoxItem name="H4" />
  <comboBoxItem name="H5" />
  <comboBoxItem name="H6" />
</toolbarComboBox>
<!--
You can remove the Font drop-down if you just want users to use Styles.
The following fonts are part of the Microsoft Core Web Fonts and are available on at least Mac OS X
and Windows
To change the default font, change the embedded style sheet in the 'style' element above.
-->
<toolbarComboBox name="Face">
  <comboBoxItem name="Arial" text="Arial" />
  <comboBoxItem name="Arial Black" text="Arial Black" />
  <comboBoxItem name="Arial Narrow" text="Arial Narrow" />
  <comboBoxItem name="Comic Sans MS" text="Comic Sans MS" />
  <comboBoxItem name="Courier New" text="Courier New" />
  <comboBoxItem name="Georgia" text="Georgia" />
  <comboBoxItem name="Impact" text="Impact" />
  <comboBoxItem name="Times New Roman" text="Times New Roman" />
  <comboBoxItem name="Trebuchet MS" text="Trebuchet MS" />
  <comboBoxItem name="Verdana" text="Verdana" />
</toolbarComboBox>
<!--
Font Size drop-down
-->
<toolbarComboBox name="Size">
  <comboBoxItem name="1" text="8pt" />
  <comboBoxItem name="2" text="10pt" />
  <comboBoxItem name="3" text="12pt" />
  <comboBoxItem name="4" text="14pt" />
  <comboBoxItem name="5" text="18pt" />
  <comboBoxItem name="6" text="24pt" />
  <comboBoxItem name="7" text="36pt" />
</toolbarComboBox>
<toolbarSeparator />
<toolbarButton name="Bold" />
<toolbarButton name="Italic" />
<toolbarButton name="Underline" />
<toolbarSeparator />
<toolbarButtonGroup name="Align" />
<toolbarSeparator />
<toolbarButtonGroup name="List" />
<toolbarButton name="DecreaseIndent" />
<toolbarButton name="IncreaseIndent" />
<toolbarSeparator />
<toolbarButton name="HighlightColor" />
<toolbarButton name="Color" />
<toolbarSeparator />
<customToolbarButton
  name="displayDialog"
  text="Display Example Dialog"
  imageURL="http://www.ephox.com/images/small_logo.gif"
  action="raiseEvent"
  value="displayDialog"
/>
</toolbar>
</toolbars>

<!--
Customize the EditLive! shortcut menu
-->
<shortcutMenu>
  <shrtMenuItem name="Undo" />
  <shrtMenuItem name="Redo" />
  <shrtMenuSeparator />

```

```
<shrtMenuItem name="Cut" />
<shrtMenuItem name="Copy" />
<shrtMenuItem name="Paste" />
<shrtMenuSeparator />
<shrtMenuItem name="Select" />
<shrtMenuSeparator />
<shrtMenuItem name="acceptChange" />
<shrtMenuItem name="rejectChange" />
<shrtMenuItem name="nextchange" />
<shrtMenuItem name="previouschange" />
<shrtMenuSeparator />
<shrtMenuItem name="Hyperlink" />
<shrtMenuItem name="RemoveHyperlink" />
<shrtMenuItem name="PropImage" />
<shrtMenuItem name="PropObject" />
<shrtMenuItem name="PropList" />
<shrtMenuItem name="PropHR" />
<shrtMenuSeparator />
<shrtMenuItem name="Split" />
<shrtMenuItem name="Merge" />
<shrtMenuItem name="tableautofit" />
<shrtMenuSeparator />
<shrtMenuItem name="PropTable" />
<shrtMenuItem name="PropRow" />
<shrtMenuItem name="PropCol" />
<shrtMenuItem name="PropCell" />
<shrtMenuSeparator />
    <shrtMenuItem name="synonyms" />
<shrtMenuItem name="EditTag" />
</shrtMenu>
</shortcutMenu>
</editlive>
```

# customToolbarItem.java

```
/*
 * Copyright (c) 2005 Ephox Corporation.
 */
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.ephox.editlive.*;
import com.ephox.editlive.common.TextEvent;
import com.ephox.editlive.common.EventListener;
import javax.jnlp.*;

/** Class loads a JFrame with a single panel, containing the
 * Ephox EditLive! Editor
 */
public class CustomToolbarItem extends JFrame implements EventListener {
    /** html content to appear in the instance of EditLive! for Java Swing */
    private static final String INITIAL_HTML = "<html><head><title>Initial Title</title></head><body><p></p></body></html>";

    /** Base class for EditLive! for Java Swing */
    private ELJBean editLiveBean = new ELJBean(INITIAL_HTML, "", 700, 570, getClass().getResource
("customItem.xml"), false);

    /** Creates JFrame and adds all class properties. Adds action listener to JButtons in JFrame
     *
     */
    public CustomToolbarItem() throws Exception {
        super("Tutorial - Custom Toolbar Items");

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // initialize EditLive! and add this class as an event listener
                editLiveBean.init();
                editLiveBean.addEditorEventListener(CustomToolbarItem.
this);
            }
        });

        this.getContentPane().setLayout(new FlowLayout());

        // Create a JPanel to hold the ELJBean
        JPanel editorHolder = new JPanel(new FlowLayout());
        editorHolder.add(editLiveBean);
        // Add the JPanel to the JFrame
        this.getContentPane().add(editorHolder);

        // Display the JFrame.
        this.setSize(new Dimension(710, 620));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    /** raiseEvent method inherited from EventListener. Every action in EditLive!
     * raises a TextEvent.
     *
     * @param e TextEvent information sent on action.
     */
}
```

```

public void raiseEvent(TextEvent e) {
    if(e.getActionCommand() == TextEvent.CUSTOM_ACTION && e.getExtraString().equals
("displayDialog")) {
        ExampleDialog exDialog = new ExampleDialog(editLiveBean);
        exDialog.setVisible(true);
    }
}

/** Sets up the application and begins its execution
 *
 * @param args the command line arguments - ignored
 */
public static void main(String args[]) throws Exception {
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(Exception e) {
        System.out.println("Unable to locate UIManager class");
        e.printStackTrace();
    }

    new CustomToolBarItem();
}

class ExampleDialog extends JFrame implements ActionListener {
    // Base class for EditLive!
    private ELJBean elBean;

    private JLabel headingLabel = new JLabel("Example Dialog");
    private JLabel infoLabel = new JLabel("Click the OK button to load the string \<b>dialog called</b>\"
into EditLive! and close this window.");
    private JButton OKButton = new JButton("OK");

    ExampleDialog(ELJBean _elBean) {
        super("Example Dialog");
        elBean = _elBean;

        OKButton.addActionListener(this);
        headingLabel.setFont(new java.awt.Font("Arial", Font.BOLD, 14));

        this.getContentPane().setLayout(new GridLayout(3, 1));
        this.getContentPane().add(headingLabel);
        this.getContentPane().add(infoLabel);

        JPanel buttonPanel = new JPanel(new FlowLayout());
        buttonPanel.add(OKButton);

        this.getContentPane().add(buttonPanel);

        // Display the JFrame.
        this.setSize(new Dimension(500, 175));
        this.setVisible(true);

        // Adding a listener to detect if the JFrame is closing, to close the application if needed.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
    }

    /** ActionListener for JButtons on the JFrame
     *
     * @param e ActionEvent sent by JButton
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == OKButton) {
            try {
                SwingUtilities.invokeAndWait(new Runnable() {

```

```
        public void run() {
            elBean.insertHTMLAtCursor("<b>dialog called</b>");

            // close window
            dispose();
        }
    });
} catch (Exception exception) {
    exception.printStackTrace();
}
}
}
```

# Using Inline Editing (Tutorial)

EditLive! provides developers with APIs for using a single instance of editor to edit multiple rich text fields occurring in a webpage. The user can simply click on the section of the page they wish to edit and an instance of EditLive! will be automatically created in it's place. The end result is a simple method of providing inline editing for a page and a significant performance enhancement over loading multiple instances of EditLive! at once.

This tutorial teaches you the key differences between implementing the standard integration of EditLive! and using the Inline Editing implementation.

## Tutorial

The [Using Inline Editing Tutorial](#) provides a step-by-step walk-through on how to integrate EditLive! into a page using the Inline Editing APIs.

## Code

The complete code views for all the associated files in the [Using Inline Editing Tutorial](#) are available [here](#).

# Using Inline Editing Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiating the Editor Tutorial](#)

## Tutorial

### Step 1. Create a Webpage Featuring Your Inline Editing Sections

Any section of your webpage you wish a user to edit (i.e. an *inline editing section*) must be nested in a HTML DIV element. For this tutorial, create a webpage named *inlineEditing.html* with two DIV elements for the user to edit (as seen below with DIV elements *content1* and *content2*).

```
<html>
  <body>
    <h1>Using Inline Editing for Content Editing</h1>

    <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and view
content.</p>

    <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

    <div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
      <h1>Inline Editing Section 1</h1>
      <p>This is the content for the <i>first</i> inline editing section</p>
    </div>

    <p>This content between the DIVs will not be editable.</p>

    <div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
      <h1>Inline Editing Section 2</h1>
      <p>This is the content for the <i>second</i> inline editing section.</p>
      <ol>
        <li>The DIV can contain any HTML content, including lists, tables, images and
links.</li>
      </ol>
    </div>
  </body>
</html>
```

### Step 2. Define a Javascript Method to be Invoked When the Page Loads

Create a Javascript method called *initializeEditableSections()* in the HEAD of the HTML document.

```
<html>
  <head>
    <script language="Javascript">
      function initializeEditableSections() {
        }
    </script>
  </head>
  <body>
    <h1>Using Inline Editing for Content Editing</h1>
```

```

content.</p>

<p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

<div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
  <h1>Inline Editing Section 1</h1>
  <p>This is the content for the <i>first</i> inline editing section</p>
</div>

<p>This content between the DIVs will not be editable.</p>

<div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
  <h1>Inline Editing Section 2</h1>
  <p>This is the content for the <i>second</i> inline editing section.</p>
  <ol>
    <li>The DIV can contain any HTML content, including lists, tables, images and
links.</li>
  </ol>
</div>
</body>
</html>

```

Specify the `initializeEditableSections()` method to be call in the BODY elements *onload* event handler.

```

<html>
  <head>
    <script language="Javascript">
      function initializeEditableSections() {
        }
    </script>
  </head>
  <body onload="initializeEditableSections();">
    <h1>Using Inline Editing for Content Editing</h1>

    <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and view
content.</p>

    <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

    <div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
      <h1>Inline Editing Section 1</h1>
      <p>This is the content for the <i>first</i> inline editing section</p>
    </div>

    <p>This content between the DIVs will not be editable.</p>

    <div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
      <h1>Inline Editing Section 2</h1>
      <p>This is the content for the <i>second</i> inline editing section.</p>
      <ol>
        <li>The DIV can contain any HTML content, including lists, tables, images and
links.</li>
      </ol>
    </div>
  </body>
</html>

```

### Step 3. Define the Required EditLive! Properties

Use the `initializeEditableSections()` javascript method to define the required properties for the editor (i.e. the [DownloadDirectory](#) and [ConfigurationFile](#) load-time properties). Ensure you also specify the `inlineEditing.js` required to implement the Inline Editing functionality.

```

<html>
  <head>

```



```

        <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript" type="
text/javascript"></script>
        <script language="Javascript">
            var editlive_js;
            function initializeEditableSections() {
                editlive_js = new EditLiveJava("editlive", "100%", "100%");
                editlive_js.setConfigurationFile("sample_eljconfig.xml");
            }
        </script>
    </head>
    <body onload="initializeEditableSections();">
        <h1>Using Inline Editing for Content Editing</h1>

        <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and view
content.</p>

        <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

        <div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
            <h1>Inline Editing Section 1</h1>
            <p>This is the content for the <i>first</i> inline editing section</p>
        </div>

        <p>This content between the DIVs will not be editable.</p>

        <div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
            <h1>Inline Editing Section 2</h1>
            <p>This is the content for the <i>second</i> inline editing section.</p>
            <ol>
                <li>The DIV can contain any HTML content, including lists, tables, images and
links.</li>
            </ol>
        </div>
    </body>
</html>

```

Note that the width and height for EditLive! have been set to 100%. This will ensure that when EditLive! is applied to one of the defined DIVs the editor will size to perfectly within the DIV.

The *editlive\_js* variable is also defined outside of the *initializeEditableSections()* function. This will ensure that any [Run Time Methods](#) to be used with EditLive! for Java will operate correctly.

## Step 4. Register the Inline Editing Sections with EditLive!

Use the [addEditableSection Method](#) to register both the *content1* and *content2* DIVs with EditLive!.

```

<html>
    <head>
        <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript" type="
text/javascript"></script>
        <script language="Javascript">
            var editlive_js;
            function initializeEditableSections() {
                editlive_js = new EditLiveJava("editlive", "100%", "100%");
                editlive_js.setConfigurationFile("sample_eljconfig.xml");
                editlive_js.addEditableSection("content1");
                editlive_js.addEditableSection("content2");
            }
        </script>
    </head>
    <body onload="initializeEditableSections();">
        <h1>Using Inline Editing for Content Editing</h1>

        <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and view
content.</p>

        <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

```

```

<div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
  <h1>Inline Editing Section 1</h1>
  <p>This is the content for the <i>first</i> inline editing section</p>
</div>

<p>This content between the DIVs will not be editable.</p>

<div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow: auto;">
  <h1>Inline Editing Section 2</h1>
  <p>This is the content for the <i>second</i> inline editing section.</p>
  <ol>
    <li>The DIV can contain any HTML content, including lists, tables, images and
links.</li>
  </ol>
</div>
</body>
</html>

```

When using the [addEditableSection Method](#) there's no need to call the [show Method](#) load-time property. Clicking in a DIV registered with EditLive! will automatically invoke [Show](#) for that DIV to create an instance of the editor.

You also don't need to populate the contents of EditLive! using either the [Body](#) or [Document](#) load-time properties. Clicking in a registered DIV will automatically populate EditLive! with the contents of that DIV.

## Step 5. Embed the Inline Editing Sections in a HTML Form

Posting the form created in this tutorial will only work if you are running this tutorial on an environment running IIS.

Finally, embed the contents of the BODY element inside a HTML FORM. This form will allow users to post the contents of their inline editing sections to the *posthandler.asp* script supplied with this tutorial.

```

<html>
  <head>
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript" type="
text/javascript"></script>
    <script language="Javascript">
      var editlive_js;
      function initializeEditableSections() {
        editlive_js = new EditLiveJava("editlive", "100%", "100%");
        editlive_js.setConfigurationFile("sample_eljconfig.xml");
        editlive_js.addEditableSection("content1");
        editlive_js.addEditableSection("content2");
      }
    </script>
  </head>
  <body onload="initializeEditableSections();">
    <form name="form1" action="posthandler.asp" method="post">
      <h1>Using Inline Editing for Content Editing</h1>

      <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and
view content.</p>

      <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

      <div id="content1" style="border: 1px solid gray; width: 600px; height: 500px; overflow:
auto;">
        <h1>Inline Editing Section 1</h1>
        <p>This is the content for the <i>first</i> inline editing section</p>
      </div>

      <p>This content between the DIVs will not be editable.</p>

      <div id="content2" style="border: 1px solid gray; width: 600px; height: 500px; overflow:
auto;">
        <h1>Inline Editing Section 2</h1>
        <p>This is the content for the <i>second</i> inline editing section.</p>
        <ol>
          <li>The DIV can contain any HTML content, including lists, tables,
images and links.</li>
        </ol>

```

```
                </div>
            </form>
        </body>
    </html>
```

When submitting a form with inline editing sections, the contents of each inline editing section will be passed to the post acceptor script as a POST argument with a name the same as the ID for the inline editing section DIV itself.

For this tutorial, when a user submits the FORM, inline editing section *content1* will be passed to *posthandler.asp* as a POST argument called *content1*. Similarly, inline editing section *content2* will be passed to *posthandler.asp* as a POST argument called *content2*.

# Using Inline Editing Code

- [inlineEditing.html](#)
- [posthandler.asp](#)

# inlineEditing.html

```
<!--
*****

inlineEditing.html --

This tutorial shows developers how to use the Editable Section
APIs available to EditLive! to use only one instance of EditLive!
for editing across several rich text sections.

Copyright © 2001-2007 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Using Inline Editing for Content Editing</title>
    <link rel="stylesheet" href="stylesheet.css">
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript" type="
text/javascript"></script>
    <script language="JavaScript" type="text/javascript">
      var editlive_js;
      function initializeEditableSections() {
        editlive_js = new EditLiveJava("editlive", "100%", "100%");
        editlive_js.setConfigurationFile("sample_eljconfig.xml");
        editlive_js.addEditableSection("content1");
        editlive_js.addEditableSection("content2");
      }
    </script>
  </head>
  <body onLoad="initializeEditableSections();">
    <form name="form1" action="posthandler.asp" method="post">
      <h1>Using Inline Editing for Content Editing</h1>

      <p>This tutorial shows how to use the EditLive! Inline Editing APIs to easily edit and
view content.</p>

      <p>Click in any section of the page with a grey border to load the selected content into
EditLive!</p>

      <div id="content1" style="border: 1px solid gray; width: 650px; height: 500px; overflow:
auto;">
        <h1>Inline Editing Section 1</h1>
        <p>This is the content for the <i>first</i> inline editing section</p>
      </div>

      <p>This content between the DIVs will not be editable.</p>

      <div id="content2" style="border: 1px solid gray; width: 650px; height: 500px; overflow:
auto;">
        <h1>Inline Editing Section 2</h1>
        <p>This is the content for the <i>second</i> inline editing section.</p>
        <ol>
          <li>The DIV can contain any HTML content, including lists, tables,
images and links.</li>
        </ol>
      </div>

      <p>Click this button to submit the form and display the contents of each inline editing
section

      <input type="submit" value="Submit Form"/></p>

      <p><b><i>Note:</i></b> The form will only submit if you are running this page from an
IIS server.</p>
    </form>
```

```
</body>  
</html>
```

# posthandler.asp

```
<%@ language="VBScript" %>
<!--
*****

postHandler.asp --

This simple script is designed to display the contents of the editable
sections passed from a HTTP form submission.

Copyright © 2001-2007 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Inline Editing Sections Content</title>
    <link rel="stylesheet" href="stylesheet.css">
  </head>
  <body>
    <p>Content from Inline Editing Section 1:</p>

    <div style="overflow: auto; border: 1px dotted gray; white-space:normal;">
    <%
      Response.Write(Request.Form("content1"))
    %>
    </div>

    <p>Content from Inline Editing Section 2:</p>

    <div style="overflow: auto; border: 1px dotted gray; white-space:normal;">
    <%
      Response.Write(Request.Form("content2"))
    %>
    </div>
  </body>
</html>
```

# Creating Plugins Utilizing Advanced APIs

EditLive!'s [Advanced APIs](#) and [plugin](#) functionality is only supported with an EditLive! [Enterprise Edition](#) license.

As demonstrated in the [Simple Plugin Tutorial](#), EditLive! provides developers with easy to use APIs for implementing plugins. Plugins can also be used to load additional Java classes for use with EditLive!. These Java classes can communicate with the editor via the [Advanced APIs](#).

This tutorial teaches you how to use plugins to declare Advanced API implementations of EditLive!.

## Tutorial

The [Creating Plugins Utilizing Advanced APIs Tutorial](#) provides a step-by-step walk-through on how to create and apply a plugin that uses the Advanced APIs to extend the functionality of EditLive!.

## Code

The complete code views for all the associated files in the [Creating Plugins Utilizing Advanced APIs Tutorial](#) are available [here](#). These codes are available for [HTML](#), [Java](#), and [XML](#).



# Creating Plugins Utilizing Advanced APIs Tutorial

## Getting Started

### Required License

EditLive!'s [Advanced APIs](#) and [plugin](#) functionality is only supported if an EditLive! [Enterprise Edition](#) license has been installed for the editor or if the user is [still](#) within their 30 day trial period.

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic knowledge of XML
- Basic knowledge of Java Swing

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Simple Plugin Tutorial](#)

## Tutorial

### Step 1. Create a Java class

Create a Java class called **AdvancedAPIPlugin**. This class will be used to extend EditLive! using the [Advanced APIs](#).

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin{

    public AdvancedAPIPlugin() {
    }
}
```

Save this class in a file called *AdvancedAPIPlugin.java*.

### Step 2. Pass EditLive! the Class

Edit the constructor for **AdvancedAPIPlugin** to require an instance of the [ELJBean](#) class. The [ELJBean](#) class is the Advanced API instance of EditLive!. When a Java class is registered with an instance EditLive! (whether through the [addJar Method](#) or [Plugin XML Elements](#)), the instance of EditLive! is passed to the constructor of the class as this [ELJBean](#) parameter.

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin{

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
    }
}
```

### Step 3. Register an EventListener with the Class

Modify **AdvancedAPIPlugin** to implement the [EventListener](#) interface. This interface is used to catch events fired by EditLive!. Register the [EventListener](#) with the instance of [ELJBean](#).

```

import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }
}

```

## Step 4. Define the raiseEvent Method for Catching Events

Events fired by EditLive! will be passed through the **raiseEvent** method. Create an instance of this method in **AdvancedAPIPlugin**.

```

import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }

    public void raiseEvent(TextEvent e) {
    }
}

```

Create a condition to catch events that have the following conditions:

- The event type is a CUSTOM\_ACTION
- The event extra type information defines a RAISE\_EVENT custom action.
- The event extra string passes "displayDialog"

These conditions will match the `<customMenuItem/>` element created in step 10 in the [Plugin XML Elements](#) used by this tutorial.

```

import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }

    public void raiseEvent(TextEvent e) {
        if (e.getActionCommand() == TextEvent.CUSTOM_ACTION) {
            if (e.getExtraInt() == TextEvent.CustomAction.RAISE_EVENT) {
                if (e.getExtraString().equals("displayDialog")) {
                }
            }
        }
    }
}

```

## Step 5. Display a Java Dialog and Insert Content into EditLive!

Handle the event caught to ensure the event is not further propagated to any other [EventListeners](#) registered with the [ELJBean](#) instance.

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }

    public void raiseEvent(TextEvent e) {
        if (e.getActionCommand() == TextEvent.CUSTOM_ACTION) {
            if (e.getExtraInt() == TextEvent.CustomAction.RAISE_EVENT) {
                if (e.getExtraString().equals("displayDialog")) {
                    e.setHandled(true);
                }
            }
        }
    }
}
```

Use the [JOptionPane](#) class to display a dialog to the user.

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }

    public void raiseEvent(TextEvent e) {
        if (e.getActionCommand() == TextEvent.CUSTOM_ACTION) {
            if (e.getExtraInt() == TextEvent.CustomAction.RAISE_EVENT) {
                if (e.getExtraString().equals("displayDialog")) {
                    e.setHandled(true);

                    JOptionPane.showMessageDialog(null, "This dialog has been generated by
the EditLive! Advanced APIs");
                }
            }
        }
    }
}
```

Use the [insertHTMLAtCursor Method](#) for [ELJBean](#) to insert a string of HTML into the EditLive! instance.

```
import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    public AdvancedAPIPlugin(ELJBean bean) {
```

```

        _bean= bean;
        bean.addEditorEventListener(this);
    }

    public void raiseEvent(TextEvent e) {
        if (e.getActionCommand() == TextEvent.CUSTOM_ACTION) {
            if (e.getExtraInt() == TextEvent.CustomAction.RAISE_EVENT) {
                if (e.getExtraString().equals("displayDialog")) {
                    e.setHandled(true);

                    JOptionPane.showMessageDialog(null, "This dialog has been generated by
the EditLive! Advanced APIs");
                    _bean.insertHTMLAtCursor("<b>Advanced API Dialog Called</b>");
                }
            }
        }
    }
}

```

## Step 6. Compile the Class into a Jar Archive

Using the instructions in the [Compiling and Running the Advanced APIs](#) article in the [Developer Guide](#) for this SDK, compile and store the **AdvancedAPIPlugin** class in a .jar archive called *AdvancedAPIPlugin.jar*.

## Step 7. Create an Instance of EditLive! for Java in a Webpage

Using the load-time properties described in the [Instantiating the Editor](#) tutorial, create an instance of EditLive!:

```

<html>
  <body>
    <h1>Adding a Plugin that Utilizes the Advanced APIs</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>

```

Save this webpage as *advancedAPIPlugin.html*.

## Step 8. Create a Plugin XML File

EditLive! loads plugins based on a [Plugin XML](#) file. For detailed information on the elements of these Plugin XML files, see the [Plugin XML Elements](#) section of the [Reference](#) guide for this SDK.

Create an .xml file called *advancedAPIPlugin.xml*. Create an empty Plugin XML structure featuring only a `<plugin>` element.

```

<?xml version="1.0" encoding="US-ASCII" ?>
<plugin>
</plugin>

```

Add the attribute **load="lazy"** to the `<plugin>` element. Using *lazy* will ensure that the plugin is only loaded when the user attempts trigger the plugin functionality via its corresponding menu item in EditLive! (see step 10).

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="lazy">
</plugin>
```

## Step 9. Reference the Java Class in the Plugin XML

Use the Plugin XML to create an `<advancedapis (Applet)>` element that references the *AdvancedAPIPlugin.jar* and its archived *AdvancedAPIPlugin.class*.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="lazy">
    <advancedapis jar="AdvancedAPIPlugin.jar" class="AdvancedAPIPlugin" />
</plugin>
```

## Step 10. Create a Custom Menu Item to be Displayed in the Plugins Menu

In the [Custom Toolbar Buttons in the Applet Tutorial](#) you've seen how custom toolbar or menu items can be easily added to EditLive! via the [Configuration File](#). This same logic can be applied to the `<menu>` element available in the Plugin XML. Any child elements found in a Plugin XML `<menu>` element will be added to the Plugin menu in EditLive!.

The event fired by a custom menu item can be processed differently by [Advanced API](#) code registered with EditLive!. For information on how Advanced API implementations can process events fired by custom menu items, see the [Configuration File Differences](#) section of the [Advanced API](#) section of the [Developer Guide](#).

Add a `<customMenuItem>` to *advancedAPIPlugin.xml* that will perform the following:

- Display a menu item titled *Display Dialog*.
- The action performed will be a *raiseEvent*.
- The string passed by this menu item will be "displayDialog" (which will then be caught by the Advanced API code seen in step 4).

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="lazy">
    <advancedapis jar="AdvancedAPIPlugin.jar" class="AdvancedAPIPlugin" />
    <menu>
        <customMenuItem name="plugin2" action="raiseEvent" value="displayDialog" text="Display
Dialog" imageURL="images/small_logo.gif"/>
    </menu>
</plugin>
```

## Step 11. Register the Plugin With EditLive!

In the *advancedAPIPlugin.html* file, add a call to the [addPlugin Method](#). Pass the URL for the *advancedAPIPlugin.xml* file as a parameter.

```
<html>
    <body>
        <h1>Adding a Plugin that Utilizes the Advanced APIs</h1>
        <!--
        Include the EditLive! JavaScript Library
        -->
        <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
        <!--
        The instance of EditLive!
        -->
        <script language="JavaScript">
            // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
            var editlive = new EditLiveJava("ELApplet", 700, 400);

            // This sets a relative or absolute path to the XML configuration file to use
            editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

            // specifying the plugin to use with EditLive!
```

```
editlive.addPlugin("../..examplePlugins/advancedAPIPlugin.xml");

// .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
// at the this location.
editlive.show();
</script>
</body>
</html>
```

# Creating Plugins Utilizing Advanced APIs Code

- [advancedAPIPlugin.html](#)
- [AdvancedAPIPlugin.java](#)
- [advancedAPIPlugin.xml](#)

# advancedAPIPlugin.html

```
<!--
*****

advancedAPIPlugin.html --

EditLive! tutorial to use only the most basic
javascript methods to instantiate the editor in a webpage.
Plugins are then used to load an Advanced API implementation
of EditLive!

Copyright © 2007 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Advanced API Plugin</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Adding a Plugin that Utilizes the Advanced APIs</h1>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // specifying the plugin to use with EditLive!
editlive.addPlugin("../../examplePlugins/advancedAPIPlugin.xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>
```



# AdvancedAPIPlugin.java

```
/* Created on 29/7/2007
 *
 * Copyright (c) 2007 Ephox Corporation.
 */

import com.ephox.editlive.*;
import com.ephox.editlive.common.*;
import javax.swing.JOptionPane;

/**
 * Class retrieves an instance of EditLive! through it's constructor. EventListener
 * is then registered with the EditLive! instance. Upon any event raised by
 * EditLive! the event is checked to see if it matches a custom menu created through
 * the advancedAPIPlugin.xml file specifying this class. If event matched custom menu, a Java swing dialog
 * appears.
 */
public class AdvancedAPIPlugin implements EventListener {

    ELJBean _bean;

    /** Constructor. Takes an instance of EditLive! and registeres an EventListener
     *
     * @param bean instance of EditLive! to be implemented in webpage
     */
    public AdvancedAPIPlugin(ELJBean bean) {
        _bean= bean;
        bean.addEditorEventListener(this);
    }

    /** raiseEvent method, called upon any EditLive! event. */
    public void raiseEvent(TextEvent e) {
        if (e.getActionCommand() == TextEvent.CUSTOM_ACTION) {
            if (e.getExtraInt() == TextEvent.CustomAction.RAISE_EVENT) {
                if (e.getExtraString().equals("displayDialog")) {
                    e.setHandled(true);

                    JOptionPane.showMessageDialog(null, "This dialog has been generated by
the EditLive! Advanced APIs");

                    _bean.insertHTMLAtCursor("<b>Advanced API Dialog Called</b>");
                }
            }
        }
    }
}
```

## advancedAPIPlugin.xml

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="lazy">
  <advancedapis jar="AdvancedAPIPlugin.jar" class="AdvancedAPIPlugin" />
  <menu>
    <customMenuItem name="plugin2" action="raiseEvent" value="displayDialog" text="Display
Dialog" imageURL="images/small_logo.gif"/>
  </menu>
</plugin>
```

# Capturing Content Before Submit

In order to use EditLive! in a web-based application, users will edit content in EditLive!, then submit the content to a server-side script which saves the information to be viewed and edited by other users.

EditLive! provides an [setAutoSubmit Method](#) which automatically copies the contents of EditLive! into a hidden form field when a form is submitted. This hidden form field, along with all other form fields, can then be processed by a server-side script to save the content on the server (in formats such as a database or simply on the server's local file system).

However, depending on how complicated a form's onsubmit function is, the AutoSubmit functionality of EditLive! may not be able to successfully write the contents of the editor to the desired hidden field.

This tutorial shows developers how to manually overwrite the onsubmit method of a HTML form to extract the contents of EditLive!.

## Tutorial

The [Capturing Content Before Submit Tutorial](#) provides a step-by-step walk-through on how to extract the contents of EditLive! and assign the contents to a form field to be processed by a server-side script.

## Code

The [complete code view](#) for all the associated files in the [Capturing Content Before Submit Tutorial](#) is also available.

# Capturing Content Before Submit Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)
- [Setting the Document in the Applet Tutorial](#)
- [Getting the Document in the Applet Tutorial](#)

## Tutorial

### Step 1. Simulating a Simple Content Editing Interface

This tutorial simulates integrating EditLive! into the editing interface of a content management system.

The following code represents a simple interface for a content management system. The webpage provides a textarea allowing an author to write some HTML content, and a Submit button to submit the content to a server-side script (myScript.asp in this example).

```
<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <p><input type="submit" value="Submit the HTML Form"/></p>
    </form>
  </body>
</html>
```

This tutorial will extend this page to integrate EditLive! as the editing interface.

Save this webpage as *capturingSubmit.html*

### Step 2. Create an Instance of EditLive! in a Webpage and Set the Document

As shown in the [Setting the Document in the Applet Tutorial](#), create an instance of EditLive! in a webpage and set the Document.

```
<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.show();
      </script>
      <p><input type="submit" value="Submit the HTML Form"/></p>
    </form>
  </body>
</html>
```

### Step 3. Set the AutoSubmit Property for EditLive! to False

For this tutorial, we're assuming the [setAutoSubmit Method](#) for EditLive! causes problems with our submit architecture. So, this property should be set to *false*.

```
<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.setAutoSubmit(false);
        editlivejava.show();
      </script>
      <p><input type="submit" value="Submit the HTML Form"/></p>
    </form>
  </body>
</html>
```

#### Step 4. Add an onsubmit Method to the HTML Form

In order to extract the contents of EditLive! when the user presses the *Submit the HTML Form* button, add a call to a Javascript method for the HTML form's **onsubmit** method.

```
<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST" onsubmit="assignFields()">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.setAutoSubmit(false);
        editlivejava.show();
      </script>
      <p><input type="submit" value="Submit the HTML Form"/></p>
    </form>
  </body>
</html>
```

#### Step 5. Write the onsubmit Method

The **onsubmit** method for HTML form needs to call the [getDocument Method](#) for EditLive!.

```
<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST" onsubmit="assignFields()">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/script>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.setAutoSubmit(false);
        editlivejava.show();
      </script>
    </form>
  </body>
</html>
```

```

        /** Function tells EditLive! to send it's HTML Document to the
getEditLiveDocument function.
        */
        function assignFields() {
            editlive.getDocument('getEditLiveDocument');
        }
    </script>
    <p><input type="submit" value="Submit the HTML Form"/></p>
</form>
</body>
</html>

```

The `GetDocument` property of `EditLive!` is an asynchronous callback property. This means that the `GetDocument` method takes an unknown time interval to call the specified method passes as a parameter. Furthermore, any code following the call to `GetDocument` will be called during this time.

In the code above, after the call to `GetDocument`, the `assignFields()` methods ends. This would then cause the HTML form to submit its contents to the `myScript.asp` server-side script.

In order to ensure the HTML form does not post until the `getEditLiveDocument` method is passed the contents of `EditLive!`, we need to ensure the `assignFields` method is treated like a boolean and returns `false`.

```

<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST" onsubmit="return assignFields()">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/scrip>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into
EditLive!</p>"));
        editlivejava.setAutoSubmit(false);
        editlivejava.show();

        /** Function tells EditLive! to send it's HTML Document to the
getEditLiveDocument function.
        */
        function assignFields() {
            editlive.getDocument('getEditLiveDocument');

            // return false to cause the form to not submit.
            return false;
        }
      </script>
      <p><input type="submit" value="Submit the HTML Form"/></p>
    </form>
  </body>
</html>

```

## Step 6. Write the `getEditLiveDocument` method

As seen in the [Getting the Document in the Applet Tutorial](#), the contents of `EditLive!` need to be copied into the `textarea`.

Now that the submit process has been canceled, we also need to manually call the HTML form's `submit` method to pass the contents of the `textarea` to the `myScript.asp` server-side script.

```

<html>
  <body>
    <form name="exampleForm" action="myScript.asp" method="POST" onsubmit="return assignFields()">
      <p><textarea id="documentContents" cols="80" rows="5"></textarea></p>
      <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"><
/scrip>
      <script>
        var editlivejava = new EditLiveJava("ELJApplet", "700", "400");
        editlivejava.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");
        editlivejava.setBody(encodeURIComponent("<p>Original <i>HTML</i> loaded into

```

```

EditLive!</p>"));
editlivejava.setAutoSubmit(false);
editlivejava.show();

/** Function tells EditLive! to send it's HTML Document to the
getEditLiveDocument function.
*/
function assignFields() {
    editlive.getDocument('getEditLiveDocument');

    // return false to cause the form to not submit.
    return false;
}

function getEditLiveDocument(src) {
    // copying source from EditLive! into textarea
    document.exampleForm.documentContents.value = src;

    // manually trigger the form submission
    document.exampleForm.submit();
}
</script>
<p><input type="submit" value="Submit the HTML Form"/></p>
</form>
</body>
</html>

```

# Capturing Content Before Submit Code

```
<!--
*****

capturingSubmit.html --

This tutorial shows developers how to extract the HTML
Document stored in EditLive! when a user submits the HTML
form.

Copyright © 2001-2006 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Capturing the Editor's HTML Document when the Form is Submitted</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Capturing the Editor's HTML Document when the Form is Submitted</h1>

    <form name="exampleForm" action="myScript.asp" method="POST" onsubmit="return assignFields()">
      <p>This tutorial shows how to extract the contents of the HTML Document in EditLive!
when the enclosing HTML form is submitted.</p>

      <!--
      The textarea used to display EditLive!'s HTML document.
      -->
      <textarea id="documentContents" cols="80" rows="5">This field has the id
'documentContents'. The contents of this textarea are passed to a server-side script called 'myScript.asp'.
Before the form is posted, the contents of EditLive! will be copied into this field.
      </textarea>

      <!--
      The instance of EditLive!
      -->
      <script language="JavaScript">
        // Create a new EditLive! instance with the name "ELApplet", a height of 400
pixels and a width of 700 pixels.
        var editlive = new EditLiveJava("ELApplet", 700, 400);

        // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava
/sample_eljconfig.xml");

        // Before sending HTML to the instance of EditLive!, this HTML must be URL
Encoded.
        // Javascript provides several URL Encoding methods, the best of which is
// 'encodeURIComponent'
editlive.setDocument(encodeURIComponent("<html><head><title>Default Document
Title</title></head><body><p>Original <i>HTML Document</i> loaded into EditLive!</p></body></html>"));

        // Disable the setAutoSubmit load-time property to ensure no errors are caused
on form submission.
editlive.setAutoSubmit(false);

        // .show is the final call and instructs the JavaScript library (editlivejava.
js) to insert a new EditLive! instance
        // at the this location.
editlive.show();
      </script>
    </body>
  </html>

```



```

getEditLiveDocument function.
    /** Function tells EditLive! to send its HTML Document to the
    */
    function assignFields() {
        editlive.getDocument('getEditLiveDocument');

        // return false to cause the form to not submit.
        return false;
    }

    function getEditLiveDocument(src) {
        // copying source from EditLive! into textarea
        document.exampleForm.documentContents.value = src;

        // inform the user the form will now submit
        alert("The contents of EditLive! have been copied into the textarea.
\nClick OK to submit the form.");

        // manually trigger the form submission
        document.exampleForm.submit();
    }
</script>

<!--
    The button for submitting the HTML form.
-->
<p><input type="submit" value="Submit the HTML Form"/></p>

</form>
</body>
</html>

```

# Simple Plugin

EditLive! provides developers with easy to use APIs for implementing plugins. Plugins are a convenient method for extending the functionality of the editor. For more information on plugins, see the [Creating and Using Plugins in the Applet](#) article in the EditLive! [Developer Guide](#).

This tutorial teaches you how to use plugins to create new menu items for EditLive! which can trigger Javascript from external Javascript files.

## Tutorial

The [Simple Plugin Tutorial](#) provides a step-by-step walk-through on how to create and apply a simple plugin to an instance of EditLive!.

## Code

The complete code views for all the associated files in the [Simple Plugin Tutorial](#) are available [here](#).

# Simple Plugin Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Custom Toolbar Buttons in the Applet Tutorial](#)

## Tutorial

### Step 1. Create an Instance of EditLive!

Using the load-time properties described in the [Instantiation Tutorial](#), create an instance of EditLive!:

```
<html>
  <body>
    <h1>Using a Simple Plugin</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>
```

Save this webpage as *simplePlugin.html*.

### Step 2. Create a Javascript File to Interact with EditLive!

Create a new Javascript file called *plugin.js*. This Javascript file should contain the following code:

- a Javascript alert. This alert will appear as soon as the Javascript file is loaded by the referencing webpage.
- a Javascript function for inserting HTML into EditLive!.

```
alert("plugin.js has loaded.");

/** This function is called to from the mockDialog.html window.
 *
 */
```

```
function insertString() {
    editlive.insertHTMLAtCursor(encodeURIComponent("<b>HTML Inserted by Plugin</b>"));
}

```

### Step 3. Create a Plugin XML File

EditLive! loads plugins based on a [Plugin XML](#) file. For detailed information on the elements of these Plugin XML files, see the [Plugin XML Elements](#) section of the [Reference](#) Guide for this SDK.

Create an *.xml* file called *simplePlugin.xml*. Create an empty Plugin XML structure featuring only a `<plugin>` element. Specify the *load* attribute as *immediate*. This will ensure that the desired Javascript is available as soon as EditLive! has loaded.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="immediate">
</plugin>

```

### Step 4. Reference the Javascript File in the Plugin XML

Use the Plugin XML to create a `<script>` element that references the *plugin.js* file.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="immediate">
  <script src="plugin.js"/>
</plugin>

```

### Step 5. Create a Custom Menu Item to be Displayed in the Plugins Menu

In the [Custom Toolbar Buttons in the Applet Tutorial](#) you've seen how custom toolbar or menu items can be easily added to EditLive! via the [Configuration File](#). This same logic can be applied to the `<menu>` element available in the Plugin XML. Any child elements found in a Plugin XML `<menu>` element will be added to the Plugin menu in EditLive!.

Add a `<customMenuItem>` to *simplePlugin.xml* that will perform the following:

- Display a menu item titled *Call to Javascript*.
- When clicked, this menu item will call the Javascript function in *plugin.js* called *insertString()*.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="immediate">
  <menu>
    <customMenuItem name="plugin1" action="raiseEvent" value="insertString" text="Call to
Javascript" imageURL="images/small_logo.gif"/>
  </menu>

  <script src="plugin.js" />
</plugin>

```

### Step 6. Register the Plugin With EditLive!

In the *simplePlugin.html* file, add a call to the [addPlugin Method](#). Pass the URL for the *simplePlugin.xml* file as a parameter.

```
<html>
  <body>
    <h1>Using a Simple Plugin</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);
    </script>
  </body>
</html>

```

```
    // This sets a relative or absolute path to the XML configuration file to use
    editlive.setConfigurationFile("../..redistributables/editlivejava/sample_eljconfig.
xml");

    // specifying the plugin to use with EditLive!
    editlive.addPlugin("../..examplePlugins/simplePlugin.xml");

    // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
    // at the this location.
    editlive.show();
        </script>
    </body>
</html>
```

# Simple Plugin Code

- [plugin.js](#)
- [simplePlugin.html](#)
- [simplePlugin.xml](#)

## plugin.js

```
alert("plugin.js has loaded.");

/** This function is called to from the mockDialog.html window.
 *
 */
function insertString() {
    editlive.insertHTMLAtCursor(encodeURIComponent("<b>HTML Inserted by Plugin</b>"));
}
```

# simplePlugin.html

```
<!--
*****

simplePlugin.html --

EditLive! tutorial to use only the most basic
javascript methods to instantiate the editor in a webpage

Copyright © 2001-2012 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Simple Plugin</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Using a Simple Plugin</h1>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // specifying the plugin to use with EditLive!
editlive.addPlugin("../../examplePlugins/simplePlugin.xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>
```



# simplePlugin.xml

```
<?xml version="1.0" encoding="US-ASCII" ?>
<plugin load="immediate">
  <menu>
    <customMenuItem name="plugin1" action="raiseEvent" value="insertString" text="Call to
Javascript" imageURL="images/small_logo.gif"/>
  </menu>

  <script src="plugin.js" />
</plugin>
```

# Optimizing Load Times

EditLive! provides developers with numerous facilities for improving the load times for the editor. For more information on the options available for increasing the load times for EditLive!, see the [Optimizing Load Times](#) article in the [Developer Guide](#) section of this SDK.

This tutorial teaches you how to utilize load-time properties and utility functions supplied by EditLive! to improve the load-time of the editor.

## Tutorial

The [Optimizing Load Times Tutorial](#) provides a step-by-step walk-through on how to improve the load times for the editor.

## Code

The complete code views for all the associated files in the [Optimizing Load Times Tutorial](#) are available [here](#).

# Optimizing Load Times Tutorial

## Getting Started

### Required Skills

The following skills are required prior to working with this tutorial:

- Basic client-side JavaScript
- Basic knowledge of XML

### Required Tutorials Completed

The following tutorials are required to be undertaken before attempting this tutorial:

- [Instantiation Tutorial](#)

## Tutorial

### Step 1. Create an Introduction Page

Create a webpage that contains a hyperlink to another webpage called *optimizingLoadTime.html*. For the purpose of this tutorial, this webpage can be seen as the introduction or sign-in page for your system implementing EditLive!

```
<html>
  <body>
    <h1>Loading the Java Virtual Machine (JVM)</h1>
    <p>This page uses the <i>quickStart</i> method to load the JVM. In the event your browser is <i>Internet Explorer</i>, the core files for EditLive! will also be downloaded to your machine via this method.</p>

    <p>Having the JVM already loaded before creating an instance of EditLive! will improve the load time for the editor.</p>

    <p>Click <a href="optimizingLoadTime.html">here</a> to load an instance of EditLive!</p>
  </body>
</html>
```

Save this page as *firstPage.html*.

### Step 2. Create an Instance of EditLive!

Using the load-time properties described in the [Instantiating the Editor tutorial](#), create an instance of EditLive!:

```
<html>
  <body>
    <h1>Using a Simple Plugin</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
      // at the this location.
```

```

        editlive.show();
    </script>
</body>
</html>

```

Save this webpage as *optimizingLoadTime.html*.

### Step 3. Use the Preload Function

Call the [preload function](#). Pass it a function that you would like to execute when EditLive! has finished loading.

```

<html>
  <body>
    <h1>Using a Simple Plugin</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
      a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      editlive.setPreload(function() {
        alert("EditLive! has finished loading.");
      });

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
      insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>

```

### Step 4. Add the ConfigurationText Load-Time Property

Delete the [setConfigurationFile Method](#) from the webpage. Replace this load-time property with the [setConfigurationText Method](#).

```

<html>
  <body>
    <h1>Using a Simple Plugin</h1>
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
      a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationText(encodeURIComponent("<?xml version=\"1.0\" ... "));

      editlive.setPreload(function() {
        alert("EditLive! has finished loading.");
      });
    </script>
  </body>
</html>

```

```
        // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
        // at the this location.
        editlive.show();
    </script>
</body>
</html>
```

Using the [setConfigurationText Method](#) to specify the EditLive! [setConfigurationFile Method](#) will cause the editor to load much faster.

# Optimizing Load Times Code

- [optimizingLoadTime.html](#)

# optimizingLoadTime.html

```
<!--
*****

optimizingLoadTime.html --

EditLive! tutorial to use specific load-time properties
to improve the load time for the editor

Copyright © 2007 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->
<html>
  <head>
    <title>Tutorial - Optimizing Load Times</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Instantiating the EditLive! Editor</h1>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationText(encodeURIComponent("<?xml version=\"1.0\" encoding=\"utf-
8\"?><!--This file customizes and configures EditLive!.TIP: this file can be dynamically generated using ASP,
JSP or PHP to achieve runtime changes to settings--><editlive> <!-- Default content for the editor -->
<document> <html> <!-- Default document header -->
<head> <!-- Specify the base URL for the editor to download all
relative images and style sheets --> <!--<base href=\"http://www.yourserver.com/cms
/\ " />--> <!-- Specify the character encoding for the editor. By
default this should be UTF-8, which will encode all special characters as numeric entities in
XHTML or as named entities in HTML --> <!--<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\" />--> <!-- Uncomment the
following line to load an external style sheet for the editor --> <!-- <link
rel=\"stylesheet\" href=\"http://www.yourserver.com/style.css\" type=\"text/css\" /> -->
<!-- Specify any embedded styles for the editor You can remove or customize the
styles below. --> <!-- <style type=\"text/css\"
> body { font-family: Verdana, Arial;
} hl { font-family: Tahoma, Arial; font-size:
24pt; font-weight: normal; color: #003366;
border-bottom: solid 1px #003366; } p.fineprint{
font-size: 8pt; text-align: center; } span.comment
{ border: solid 1px #FFFF00; background-color:
#FFFFCC; } </style> --> <
/head> <!-- Default document body. Add content here if you want this to be
the default when the editor loads, although this is better done at runtime. --
> <body> </body> </html> </document> <!-- Add your Ephox-provided
license key here --> <ephoxLicenses> <license accountID=\"
BB56B8DD47EF\" activationURL=\"http://www.ephox.com/elregister/el2/activate.asp\"
domain=\"LOCALHOST\" expiration=\"NEVER\" forceActive=\"false\" key=\"6FFF-
9765-8052-7AA5\" licensee=\"For Evaluation Only\" product=\"EditLive!\"
release=\"6.0\" seats=\"\" type=\"Evaluation License\" eqEditor=\"
true\" /> </ephoxLicenses> <!-- Specify the location of the spell checker
and thesaurus. If no spellcheck or thesaurus jars are specified, the location for these jars is
automatically generated based on the specified DownloadDirectory load-time property and the user's
locale. --> <!-- <spellCheck jar=\"../../redistributables/editlivejava/dictionaries/en_us_4_0.jar\"

```

```

useNotModified="\false\ "> </spellCheck> <thesaurus jar="\../redistributables/editlivejava/thesaurus
/thes_am_6_0.jar\ " useNotModified="\false\ "/> --> <!-- Specify HTML filter settings -->
<htmlFilter outputXHTML="\true\ " outputXML="\false\ " indentContent="\false\ "
logicalEmphasis="\true\ " quoteMarks="\false\ " removeFontTags="\false\ " uppercaseTags="\
false\ " uppercaseAttributes="\false\ " wrapLength="\0\ "> </htmlFilter> <!-- Specify
settings for the Design (WYSIWYG) view(s) of the editor. Set tabPlacement="\off\ " to disable the tabs. --
> <wysiwygEditor tabPlacement="\bottom\ " brOnEnter="\false\ "
showDocumentNavigator="\false\ " disableInlineImageResizing="\false\ "
disableInlineTableResizing="\false\ " enableTrackChanges="\false\ " > <!-- Define Custom Tags
actions --> <!-- <customTags> <doubleClickActions> <action.../> <
/doubleClickActions> </customTags> --> <!-- Define additional symbols for the symbol dialog
here --> <!-- <symbols></symbols> --> </wysiwygEditor> <!-- Specify settings for
the Source (code) view of the editor --> <sourceEditor showBodyOnly="\false\ "/> <!-- Specify
options for content that EditLive has detected has been pasted from Microsoft Word --> <wordImport
styleOption="\merge_inline_styles\ "/> <!-- Specify options for content that EditLive has detected
has been pasted from another HTML document --> <htmlImport styleOption="\merge_inline_styles\ "
/> <mediaSettings> <!-- Specify HTTP upload settings 'base' is the
base URL used to update the 'src' attributes of any local files in the HTML source 'href' is your
server-side script for handling multipart-formdata uploads from ELJ The httpUploadData element
specifies any additional fields to post with the image data --> <!--
<httpUpload base="\http://www.yourserver.com/userfiles/\ " href="\http://www.yourserver.
com/scripts/upload.jsp"> <httpUploadData name="\hello\ " data="\world\ "/> <
/httpUpload> --> <images allowLocalImages="\true\ " allowUserSpecified="\true\ "> <!--
-- The list of images which appear in the Insert Image dialog. TIP: Dynamically generate
this from your database or repository to achieve an easy image library. -->
<imageList <image name="\Ephox EditLive!\ " description="\Ephox EditLive!
Logo\ " alt="\Ephox EditLive! Logo\ " src="\http://www.ephox.com/product
/editliveforjava/demos/startcms/images/eljlogo.jpg\ " title="\Ephox EditLive!\ "
/> <image name="\iMac\ " alt="\iMac
Computer\ " description="\iMac Computer\ " title="\
iMac\ " border="\0\ " src="\http://www.ephox.com/product
/editliveforjava/demos/startcms/images/newimac.gif\ " /> <image name="\Apple
Computer\ " alt="\Apple Computer\ " title="\Apple
Computer\ " description="\Picture of a new Apple Computer\ " border="\
0\ " src="\http://www.ephox.com/product/editliveforjava/demos/startcms/images/applecomp.jpg\ "
/> <image name="\IBM Thinkpad\ " alt="\IBM
Thinkpad\ " border="\0\ " title="\IBM Thinkpad\ "
description="\Picture of a new IBM Thinkpad\ " src="\http://www.ephox.com/product
/editliveforjava/demos/startcms/images/ibm_thinkpad.gif\ " />
</imageList> </images> <multimedia> <types> <type name="\Macromedia
Flash\ " type="\application/x-shockwave-flash\ " extension="\swf\ " allowCustomParams="\true\ " urlParam="\movie\ "
> <param name="\quality\ " />
<param name="\bgcolor\ " /> </type> <type name="\QuickTime Movie\ " type="\video
/quicktime\ " extension="\mov\ " allowCustomParams="\true\ ">
<param name="\autoplay\ " /> <param name="\bgcolor\ "
/> <param name="\cache\ " /> <param name="\controller\ "
/> <param name="\correction\ " /> <param name="\dontflattenwhensaving\ "
/> <param name="\enablejavascript\ " /> <param name="\endtime\ "
/> <param name="\fov\ " /> <param name="\height\ " />
<param name="\href\ " /> <param name="\kioskmode\ " /> <param name="\loop\ "
/> <param name="\movieid\ " /> <param name="\moviename\ "
/> <param name="\node\ " /> <param name="\pan\ " />
<param name="\playeveryframe\ " /> <param name="\qtsrcchokespeed\ " />
<param name="\scale\ " /> <param name="\starttime\ " /> <param name="\
target\ " /> <param name="\targetcache\ " /> <param name="\tilt\ "
/> <param name="\urlsubstitute\ " /> <param name="\volume\ "
/> </type> <type name="\Window Media\ " type="\application/x-mplayer2\ "
extension="\asf\ " allowCustomParams="\true\ " urlParam="\fileName\ "> <param name="\
animationAtStart\ " /> <param name="\autoStart\ " /> <param name="\
showControls\ " /> <param name="\clickToPlay\ " /> <param name="\
transparentAtStart\ " /> </type> <type name="\Window Media (Streaming)\ " type="\
application/x-mplayer2\ " extension="\asx\ " allowCustomParams="\true\ " urlParam="\fileName\ ">
<param name="\animationAtStart\ " /> <param name="\autoStart\ " />
name="\showControls\ " /> <param name="\clickToPlay\ " /> <param name="\
transparentAtStart\ " /> </type> <type name="\WAV Audio\ " type="\application/x-
mplayer2\ " extension="\wav\ " allowCustomParams="\true\ " /> <type name="\MP3 Audio\ " type="\
application/x-mplayer2\ " extension="\mp3\ " allowCustomParams="\true\ " /> <type name="\AVI\ "
type="\application/x-mplayer2\ " extension="\avi\ " allowCustomParams="\true\ " />
</types> </multimedia> </mediaSettings> <hyperlinks>
<hyperlinkList <hyperlink href="\http://www.ephox.com\ " description="\Ephox Web site\ "
/> <hyperlink href="\http://www.apple.com\ " description="\Apple Computer Web site\ " />

```



```

<hyperlink href="\http://www.sun.com\" description="\Sun Microsystems Web site\" /> <
/hyperlinkList> <mailtoList> <mailtoLink href="\mailto:info@ephox.com\" description="\
Ephox information\" /> </mailtoList> </hyperlinks> <!-- Customize the EditLive!
menus Note: you must display some sort of Ephox copyright statement within your application, only
remove the About menu (by setting showAboutMenu="\false\") if you have correctly attributed Ephox's
copyright in the appropriate place(s) within your application. --> <menuBar showAboutMenu="\true\"
> <menu name="\ephox_filemenu\"> <menuItem name="\New\"/> <menuItem name="\
Open\"/> <menuSeparator/> <menuItem name="\Save\"/> <menuItem name="\SaveAs\"
/> <menuSeparator/> <menuItem name="\Print\"/> </menu> <menuItem name="\
name="\ephox_editmenu\"> <menuItem name="\Undo\"/> <menuItem name="\Redo\"/>
<menuSeparator/> <menuItem name="\Cut\"/> <menuItem name="\Copy\"/> <menuItem
name="\Paste\"/> <menuItem name="\PasteSpecial\"/> <menuSeparator/> <menuItem
name="\Select\"/> <menuItem name="\SelectAll\"/> <menuSeparator/> <menuItem
name="\Find\"/> <menuSeparator/> </menu> <menu name="\ephox_viewmenu\"
> <menuItemGroup name="\SourceView\"/> <menuSeparator/> <menuItem name="\
Popout\"/> <menuSeparator/> <menuItem name="\showDocumentNavigator\"/>
<menuSeparator/> <menuItem name="\ParagraphMarker\"/> </menu> <menu
name="\ephox_insertmenu\"> <menuItem name="\HLink\"/> <menuItem name="\Bookmark\"
/> <menuItem name="\RemoveHyperlink\" /> <menuSeparator/> <menuItem name="\
ImageServer\"/> <menuItem name="\InsertObject\"/> <menuSeparator/> <menuItem
name="\Symbol\"/> <menuItem name="\HRule\"/> <menuSeparator/> <menuItem
name="\DateTime\"/> <menuSeparator/> <menuItem name="\insertcomment\"
/> </menu> <menu name="\ephox_formatmenu\"> <submenu name="\Style\"
/> <submenu name="\Face\"/> <submenu name="\Size\"/> <menuSeparator
/> <menuItem name="\Bold\"/> <menuItem name="\Italic\"/> <menuItem name="\
Underline\"/> <menuSeparator/> <menuItemGroup name="\Align\"/> <menuSeparator
/> <menuItemGroup name="\List\"/> <menuItem name="\DecreaseIndent\"/>
<menuItem name="\IncreaseIndent\"/> <menuItem name="\PropList\"/> <menuSeparator
/> <menuItemGroup name="\Script\"/> <menuItem name="\Strike\"/> <menuSeparator
/> <menuItem name="\RemoveFormatting\"/> <menuItem name="\FormatPainter\"/>
</menu> <menu name="\ephox_toolsmenu\"> <menuItem name="\Spelling\"/>
<menuItem name="\BackgroundSpellChecking\"/> <menuItem name="\thesaurus\"/> <menuSeparator
/> <menuItem name="\Accessibility\"/> <menuSeparator/> <menuItem name="\
WordCount\"/> </menu> <menu name="\ephox_tablemenu\"> <menuItem name="\
InsTable\"/> <menuItem name="\InsRowCol\"/> <menuSeparator/> <menuItem name="\
DelRow\"/> <menuItem name="\DelCol\"/> <menuSeparator/> <menuItem name="\
Split\"/> <menuItem name="\Merge\"/> <menuItem name="\tableautofit\"/>
<menuSeparator/> <menuItem name="\PropCell\"/> <menuItem name="\PropRow\"/>
<menuItem name="\PropCol\"/> <menuItem name="\PropTable\"/> <menuSeparator/>
<menuItem name="\Gridlines\"/> </menu> <menu name="\ephox_formmenu\">
<menuItem name="\InsForm\"/> <menuSeparator/> <menuItem name="\
InsTextField\"/> <menuItem name="\InsPasswordField\"/> <menuItem name="\
InsHiddenField\"/> <menuItem name="\InsFileField\"/> <menuItem name="\
InsButtonField\"/> <menuItem name="\InsSubmitField\"/> <menuItem name="\
InsResetField\"/> <menuItem name="\InsCheckboxField\"/> <menuItem name="\
InsRadioField\"/> <menuItem name="\InsTextAreaField\"/> <menuItem name="\
InsSelectField\"/> <menuItem name="\InsImageField\"/> </menu> <menu
name="\ephox_trackchangesmenu\"> <menuItem name="\enabletrackchanges\" />
<menuSeparator /> <menuItem name="\acceptChange\" /> <menuItem name="\
rejectChange\" /> <menuSeparator /> <menuItem name="\previousChange\"
/> <menuItem name="\nextChange\" /> <menuSeparator />
<menuItem name="\acceptAllChanges\" /> <menuItem name="\rejectAllChanges\"
/> <menuSeparator /> <menuItem name="\showTrackChangesDialog\"
/> <menuSeparator /> <menuItem name="\setUsername\" /> </menu> <
/menuBar> <!-- Customize the EditLive! toolbars --> <toolbars> <toolbar name="\Command\"
> <toolbarButton name="\Print\"/> <toolbarSeparator/> <toolbarButton name="\
Spelling\"/> <toolbarButton name="\Find\"/> <toolbarSeparator/>
<toolbarButton name="\Cut\"/> <toolbarButton name="\Copy\"/> <toolbarButton name="\Paste\"
/> <toolbarButton name="\FormatPainter\" /> <toolbarSeparator/>
<toolbarButton name="\Undo\"/> <toolbarButton name="\Redo\"/> <toolbarSeparator
/> <toolbarButton name="\HLink\"/> <toolbarButton name="\ImageServer\"
/> <toolbarButton name="\insertequation\"/> <toolbarSeparator
/> <toolbarButton name="\InsTableWizard\"/> <toolbarButton name="\InsRow\"
/> <toolbarButton name="\InsCol\"/> <toolbarButton name="\DelRow\"/>
<toolbarButton name="\DelCol\"/> <toolbarSeparator/> <toolbarButton name="\
enableTrackChanges\"/> <toolbarButton name="\acceptChange\"/> <toolbarButton name="\
rejectchange\"/> <toolbarButton name="\previouschange\"/> <toolbarButton name="\
nextchange\"/> <toolbarSeparator/> <toolbarButton name="\ParagraphMarker\"
/> <toolbarSeparator/> <toolbarButton name="\Popout\"/> </toolbar>
<toolbar name="\Format\"> <!-- Styles from any embedded or external stylesheets will
also be automatically added to the Styles drop-down --> <toolbarComboBox name="\Style\"

```

```

> <comboBoxItem name=\P\"/> <comboBoxItem name=\H1\"/>
<comboBoxItem name=\H2\"/> <comboBoxItem name=\H3\"/> <comboBoxItem name=\H4\"
/> <comboBoxItem name=\H5\"/> <comboBoxItem name=\H6\"/> <
/toolbarComboBox> <!-- You can remove the Font drop-down if you just want users to use
Styles. The following fonts are part of the Microsoft Core Web Fonts and are available on at least
Mac OS X and Windows To change the default font, change the embedded style sheet in the 'style'
element above. --> <toolbarComboBox name=\Face\"> <comboBoxItem name=\
Arial\" text=\Arial\"/> <comboBoxItem name=\Arial Black\" text=\Arial Black\"
/> <comboBoxItem name=\Arial Narrow\" text=\Arial Narrow\"/> <comboBoxItem
name=\Comic Sans MS\" text=\Comic Sans MS\"/> <comboBoxItem name=\Courier New\" text=\
Courier New\"/> <comboBoxItem name=\Georgia\" text=\Georgia\"/> <comboBoxItem
name=\Impact\" text=\Impact\"/> <comboBoxItem name=\Times New Roman\" text=\Times New Roman\"
/> <comboBoxItem name=\Trebuchet MS\" text=\Trebuchet MS\"/> <comboBoxItem
name=\Verdana\" text=\Verdana\"/> </toolbarComboBox> <!-- Font Size drop-
down --> <toolbarComboBox name=\Size\"> <comboBoxItem name=\1\" text=\
8pt\"/> <comboBoxItem name=\2\" text=\10pt\"/> <comboBoxItem name=\3\" text=\
12pt\"/> <comboBoxItem name=\4\" text=\14pt\"/> <comboBoxItem name=\5\" text=\
18pt\"/> <comboBoxItem name=\6\" text=\24pt\"/> <comboBoxItem name=\7\" text=\
36pt\"/> </toolbarComboBox> <toolbarSeparator/> <toolbarButton name=\Bold\"
/> <toolbarButton name=\Italic\"/> <toolbarButton name=\Underline\"/>
<toolbarSeparator/> <toolbarButtonGroup name=\Align\"/> <toolbarSeparator/>
<toolbarButtonGroup name=\List\"/> <toolbarButton name=\DecreaseIndent\"/>
<toolbarButton name=\IncreaseIndent\"/> <toolbarSeparator/> <toolbarButton name=\
HighlightColor\"/> <toolbarButton name=\Color\"/> </toolbar> </toolbars> <!--
Customize the EditLive! shortcut menu --> <shortcutMenu> <shrtMenuItem
name=\Undo\"/> <shrtMenuItem name=\Redo\"/> <shrtMenuItemSeparator/>
<shrtMenuItem name=\Cut\"/> <shrtMenuItem name=\Copy\"/> <shrtMenuItem name=\Paste\"
/> <shrtMenuItemSeparator/> <shrtMenuItem name=\Select\"/> <shrtMenuItemSeparator
/> <shrtMenuItem name=\acceptChange\"/> <shrtMenuItem name=\rejectChange\"/>
<shrtMenuItem name=\nextchange\"/> <shrtMenuItem name=\previouschange\"/>
<shrtMenuItemSeparator/> <shrtMenuItem name=\Hyperlink\"/> <shrtMenuItem name=\
RemoveHyperlink\"/> <shrtMenuItem name=\PropImage\"/> <shrtMenuItem name=\PropObject\"
/> <shrtMenuItem name=\PropList\"/> <shrtMenuItem name=\PropHR\"/>
<shrtMenuItemSeparator/> <shrtMenuItem name=\Split\"/> <shrtMenuItem name=\Merge\"
/> <shrtMenuItem name=\tableautofit\"/> <shrtMenuItemSeparator/> <shrtMenuItem
name=\PropTable\"/> <shrtMenuItem name=\PropRow\"/> <shrtMenuItem name=\PropCol\"
/> <shrtMenuItem name=\PropCell\"/> <shrtMenuItemSeparator/>
<shrtMenuItem name=\synonyms\"/> <shrtMenuItem name=\EditTag\"/> </shrtMenuItem> <
/shortcutMenu></editlive>));

editlive.setPreload(function() {
    alert("EditLive! has finished loading.");
});

// .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
// at the this location.
editlive.show();
</script>
</body>
</html>

```

# Examples

- [Introduction](#)
- [Basic ASP Example](#)
  - [Basic ASP Example Documentation](#)
  - [Basic ASP Example Code](#)
- [Basic ASP.NET Example](#)
  - [Basic ASP.NET Example Documentation](#)
  - [Basic ASP.NET Example Code](#)
- [Inline Editing ASP.NET Example](#)
  - [Inline Editing in ASP.NET Example Documentation](#)
  - [Inline Editing in ASP.NET Example Code](#)
- [Basic ColdFusion Example](#)
  - [Basic ColdFusion Example Documentation](#)
  - [Basic ColdFusion Example Code](#)
- [Basic PHP Example](#)
  - [Basic PHP Example Documentation](#)
  - [Basic PHP Code](#)
- [ColdFusion in MySQL Example](#)
  - [MySQL in ColdFusion Example Documentation](#)
  - [MySQL in ColdFusion Code](#)
    - [add.cfm](#)
    - [delete.cfm](#)
    - [edit.cfm](#)
    - [start.cfm](#)
    - [view.cfm](#)
    - [xt\\_add.cfm](#)
    - [xt\\_delete.cfm](#)
    - [xt\\_edit.cfm](#)
- [MySQL in PHP Example](#)
  - [MySQL in PHP Example Documentation](#)
  - [MySQL in PHP Code](#)
    - [add.php](#)
    - [delete.php](#)
    - [edit.php](#)
    - [i\\_database.php](#)
    - [start.php](#)
    - [view.php](#)
    - [xt\\_add.php](#)
    - [xt\\_delete.php](#)
    - [xt\\_edit.php](#)
- [Automated Plugin Loading PHP Example](#)
  - [Automated Plugin Loading PHP Documentation](#)
  - [Automated Plugin Loading PHP Code](#)
    - [example.php](#)
    - [pluginLoader.php](#)

# Introduction

This SDK provides developers with examples of using EditLive! in Web based applications.

Each example comes packaged with documentation to explain the processes involved in creating the specified application. Also packaged is the code view for each example to allow users to walk through the logic involved behind each application.

## Requirements

Each example requires this SDK to be deployed on a specific type of web server (e.g. Java Server, IIS Server). For more information on deploying EditLive! on specified web servers, see the EditLive! [Install Guide](#).

# Basic ASP Example

## Documentation

The [Basic ASP Example Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code view for all the associated files in the Basic ASP example are available [here](#).

# Basic ASP Example Documentation

This page provides information on how to instantiate Tiny EditLive! within a web page using Active Server Pages (ASP) code. This is intended as a basic sample only. Its purpose is only to demonstrate how Tiny EditLive! can be placed into a web page.

## Getting Started

### System Requirements

If you are running Microsoft Windows XP Server or Professional, ensure that the following components are installed on your server:

- Microsoft Internet Information Server 5.1

If you are running Microsoft Windows 2003, ensure that the following components are installed on your server:

- Microsoft Internet Information Server 5.0

### Required Skills

The following skills are required prior to working with this example:

- Developing Web-based forms in HTML
- Processing Web-based forms in Active Server Pages
- Basic client-side JavaScript

## Overview

In this example, EditLive! is embedded into a Web page using ASP code. A database is not required for this basic sample. You cannot perform any saving of document content in this basic sample due to the lack of server-side processing in the example code and the absence of a database. This sample is intended to show only how to create an instance of EditLive! within a Web page through the use of ASP code.

This sample demonstrates how to perform the following with EditLive! and ASP:

- Embed an instance of EditLive! in a Web page using ASP code.
- Set variables affecting the appearance of EditLive! within the Web page.

## EditLive! in basicspsample.asp

To embed EditLive! within a Web page several steps are required. Each of these steps is explained here and code samples are provided.

1. Include the *editlivejava.asp* file.

```
<!-- #include file="../../redistributables/editlivejava/editlivejava.asp" -->
```

The *editlivejava.asp* provides an interface to the EditLive! JavaScript file *editlivejava.js*. *Editlivejava.asp* includes methods which will be used in creating an instance of EditLive!. This file can be found in the *redistributables/editlivejava* subdirectory of the EditLive! directory.

2. Create a HTML form to place an instance of EditLive! in.

```
<form name="form1" method="POST">
```

3. Create the EditLive! global object.

```
<% 'Declare a global EditLive! object
Dim eljglobal
Set eljglobal = New EditLiveGlobal
'Set the download directory to where EditLive! can be found
eljglobal.DownloadDirectory = "../../redistributables/editlivejava"
eljglobal.LocalDeployment="false"
eljglobal.Init()
```

Embedding EditLive! in a Web page requires the creation of two objects within a section of Visual Basic script. The first of these objects is an EditLive! global object. This object stores where the download directory of EditLive! can be found. In this case it is the *redistributables/editlivejava* subdirectory of the EditLive! directory. Noted also is the location to download the Java Run Time Environment (JRE) if required, but this is dependent on the **LocalDeployment** load-time property. The object is then initialized by calling its **Init()** method.

For more information on the properties of this object see [EditLive! for Java Instantiation Reference](#). For more information on [Deploying Tiny EditLive!](#) please see the [EditLive! for Java SDK](#).

#### 4. Creating the EditLive! instance object.

```
'Declare an instance of EditLive!  
Dim editlivejava1  
Set editlivejava1 = New EditLive  
'Set the properties for the instance of EditLive!  
editlivejava1.Name = "ELJApplet1"  
editlivejava1.Width = 600  
editlivejava1.Height = 400  
' Specify the location of the XML configuration file for EditLive!  
editlivejava1.ConfigurationFile = "sample_eljconfig.xml"  
' Specify the initial content for EditLive!  
editlivejava1.Body = "<p> Document contents for EditLive! </p>"  
' Show the editor applet  
editlivejava1.Show()  
%>
```

This section of code creates an instance of EditLive! within the page and sets variables which affect how EditLive! will be presented within the page. After each of the properties have been set the **Show()** method is called. This method causes the instance of EditLive! just created to be displayed in the Web page. It should be noted that this is the closing portion of the Visual Basic script started in step three.

### Summary

EditLive! may be quickly and easily placed within an Active Server Page using some simple Visual Basic scripting. EditLive! may have its size and configuration easily customized using the properties as listed in step four of the above procedure.

# Basic ASP Example Code

```
<html>
  <body>
    <!-- #include file="../../redistributables/editlivejava/editlivejava.asp" -->
    <form name="form1" method="POST">
      <% 'Declare a global EditLive! object
        Dim eljglobal
        Set eljglobal = New EditLiveGlobal
        'Set the download directory to where EditLive! can be found
        eljglobal.DownloadDirectory = "../../redistributables/editlivejava"
        eljglobal.LocalDeployment="false"
        eljglobal.Init()

        'Declare an instance of EditLive!
        Dim editlivejava1
        Set editlivejava1 = New EditLive
        'Set the properties for the instance of EditLive!
        editlivejava1.Name = "ELJApplet1"
        editlivejava1.Width = 600
        editlivejava1.Height = 400
        ' Specify the location of the XML configuration file for EditLive!
        editlivejava1.ConfigurationFile = "sample_eljconfig.xml"
        ' Specify the initial content for EditLive!
        editlivejava1.Body = "<p> Document contents for EditLive! </p>"
        ' Show the editor applet
        editlivejava1.Show()
      %>
    </form>
  </body>
</html>
```



# Basic ASP.NET Example

## Documentation

The [Basic ASP.NET Example Documentation](#) contains instructions on installing and configuring EditLive! for Java in ASP.NET applications.

## Code

The complete code view for all the associated files in the Basic ASP.NET example are available [here](#).

# Basic ASP.NET Example Documentation

This page provides information on how to instantiate EditLive! within a web page using Active Server Pages in the .NET environment. This is intended as a basic sample only. Its purpose is only to demonstrate how EditLive! can be placed into a web page.

## Getting Started

### System Requirements

- Microsoft IIS 6.0 Web Server with ASP.NET 3.0 or later installed
- The EditLive! ASP.NET Server Control

### Required Skills

The following skills are required prior to working with this sample:

- Developing Web-based forms in HTML
- Developing ASP.NET Web Forms in C#

### Installing the EditLive! Server Control Component

For information on how to install the EditLive! ASP.NET Server Control please see the [EditLive! ASP.NET Installation Guide](#) document in the SDK. This example assumes that you have read the document. Furthermore, it is assumed that you have set up a Web project for development and included the appropriate files for development with the EditLive! ASP.NET Server Control.

### Source Code

This documentation includes a copy of the full source code needed for this example. You will find the completed source code in a file called BasicSample.aspx in the directory *EDITLIVE\_INSTALL/webfolder/aspnet/basic* where *EDITLIVE\_INSTALL* is the location where EditLive! is installed. In order to run this example add the BasicSample.aspx file to an ASP.NET Web project which includes a reference to the EditLiveJavaControl.dll file. Then example will then be available to browse to in the Web project concerned.

## Instantiating EditLive! in an ASP.NET Page

1. Add a reference to the *EditLiveJavaControl.dll* in the relevant project. Also ensure that the EditLive! source files and libraries are present on the Web server. For more information on how to do this please see the [ASP.NET Installation Guide](#) document.
2. Create a new Web Form.
3. Using the **Register** directive create a prefix for the EditLive! Server Control and register the name space. In this case we will be using the prefix *elj*. The **Namespace** and **Assembly** values are both *EditLiveJavaControl*.

```
<%@ Register assembly="EditLiveJavaControl" namespace="EditLiveJavaControl" tagprefix="elj" %>
```

4. Declare a tag which represents the EditLive! Server Control and set the **runat** property to **server**.

```
<form id="form1" runat="server">
...
<elj:EditLiveJava runat="server" ... />
...
</form>
```

5. Specify the **ID** property for the control instance.

```
<form id="form1" runat="server">
...
<elj:EditLiveJava runat="server" ID="EditLiveJava1" ... />
...
</form>
```

6. Specify the download directory for the instance of EditLive!. This is the location of *editlivejava.jar* and associated files. This defaults to */editlivejava*.

```
<form id="form1" runat="server">
...
<elj:EditLiveJava runat="server" id="EditLiveJava1"
```

```
DownloadDirectory="../../redistributables/editlivejava"
... />
...
</form>
```

7. Specify the XML configuration file for the instance of EditLive!.

```
<form id="form1" runat="server">
...
<elj:EditLiveJava runat="server" id="EditLiveJava1"
DownloadDirectory="../../redistributables/editlivejava"
ConfigurationFile="../../redistributables/editlivejava/sample_eljconfig.xml"
... />
...
</form>
```

8. Specify the initial contents of EditLive!. The content must be HTML-encoded.

```
<form id="form1" runat="server">
...
<elj:EditLiveJava runat="server" id="EditLiveJava1"
DownloadDirectory="../../redistributables/editlivejava"
ConfigurationFile="../../redistributables/editlivejava/sample_eljconfig.xml"
Content="&lt;p&gt;Default editor content&lt;/p&gt;&lt;p&gt;More content&lt;/p&gt;"
/>
...
</form>
```

The set of properties used in this example consists of the basic properties required to be set when creating an instance of EditLive! in an ASP.NET page. For more information see the [ASP.NET Installation Guide](#) and the EditLive! for Java [Load Time Methods](#).

## Retrieving Content from EditLive! for Java in an ASP.NET Page

The code listed in the various steps of the previous section details how to instantiate EditLive! within an ASP.NET page and specify the initial content. The following describes how to retrieve the content from the control in a server-side event handler.

1. Include a button on the page which will get the contents of EditLive! when clicked. This calls the function written in step 3 of this section.

```
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />
```

2. Include a textbox on the page to display the content of EditLive! in once it has been retrieved.

```
<asp:TextBox ID="TextBox1" runat="server" Height="96px" TextMode="MultiLine" Width="252px" />
```

3. Write a server-side event handler to retrieve the content of EditLive! when the button is clicked. We simply read the Content property from the EditLiveJava control into the Text property of the textbox.

```
protected void Button1_Click(object sender, EventArgs e) {
    TextBox1.Text = EditLiveJava1.Content;
}
```

By default ASP.NET forms cannot post values containing HTML. In order to successfully post EditLive!'s content, you will need to ensure the page directive contains the `validateRequest="false"` parameter.

## Summary

With the EditLive! ASP.NET Server Control, EditLive! can be quickly and easily integrated into an ASP.NET page. The server control allows for seamless integration with the ASP.NET Architecture, meaning that EditLive!'s content can be easily submitted to an ASP.NET server page and retrieved.

To run this example, change any required settings in the *basicexample.aspx* file. This file can be found in *EDITLIVE\_INSTALL/webfolder/aspnet/basic/*, where *EDITLIVE\_INSTALL* is the directory where you have installed EditLive!.

By default ASP.NET forms cannot post values containing HTML. In order to successfully post EditLive!'s content, you will need to ensure the page directive contains the `validateRequest="false"` parameter.

In ASP.NET 4.0, you also need to add the following to your web.config, in the <system.web> section:

```
<httpRuntime requestValidationMode="2.0" />
```

# Basic ASP.NET Example Code

## ASPX Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="BasicSample.aspx.cs" Inherits="basic_BasicSample"
ValidateRequest="false" %>
<%@ Register assembly="EditLiveJavaControl" namespace="EditLiveJavaControl" tagprefix="elj" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Basic Example</title>
</head>
<body>
  <form id="form1" runat="server">
    <elj:EditLiveJava ID="EditLiveJava1" runat="server"
      DownloadDirectory="../../redistributables/editlivejava"
      ConfigurationFile="../../redistributables/editlivejava/sample_eljconfig.xml"
      Content="&lt;p&gt;Default editor content&lt;/p&gt;&lt;p&gt;More content&lt;/p&gt;"
    />
    <br />
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />
    <br />
    <asp:TextBox ID="TextBox1" runat="server" Height="96px" TextMode="MultiLine" Width="252px" />
    <br />
  </form>
</body>
</html>
```

## C# Code-Behind

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class basic_BasicSample : System.Web.UI.Page {
    protected void Button1_Click(object sender, EventArgs e) {
        TextBox1.Text = EditLiveJava1.Content;
    }
}
```

# Inline Editing ASP.NET Example

## Documentation

The [Inline Editing in ASP.NET Example Documentation](#) extends from the [Basic ASP.NET Example](#) to describe how to implement EditLive! for Java [Inline Editing](#) into your ASP.NET application.

## Code

The complete code view for all the associated files in the Inline Editing in ASP.NET example is available [here](#).

# Inline Editing in ASP.NET Example Documentation

This page provides information on how to instantiate EditLive! [Inline Editing](#) within a web page using Active Server Pages in the .NET environment. This is intended as a basic sample only.

## Getting Started

### System Requirements

- Microsoft IIS 6.0 Web Server with ASP.NET 3.0 installed
- The EditLive! ASP.NET Server Control

### Required Skills

The following skills are required prior to working with this sample:

- Developing Web-based forms in HTML
- Developing ASP.NET Web Forms

### Source Code

This documentation includes a copy of the full source code needed for this example. You will find the completed source code in a file called *InlineEditingExample.aspx* in the directory *EDITLIVE\_INSTALL/webfolder/aspnet/inlineEditing*, where *EDITLIVE\_INSTALL* is the location where EditLive! is installed. In order to run this example add the *InlineEditingExample.aspx* file to an ASP.NET Web project which includes a reference to the *EditLiveJavaControl.dll* file.

## Instantiating EditLive! in InlineEditingExample.aspx

For more information on ASP.NET Web Forms, please consult the Microsoft .NET SDK.

1. Perform the same basic steps as seen in the [Basic ASP.NET Example](#) to reference the EditLive! ASP.NET Control and create an instance of the editor.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="InlineEditingExample.aspx.cs" Inherits="
inlineEditing_InlineEditingExample" ValidateRequest="false" %>
<%@ Register assembly="EditLiveJavaControl" namespace="EditLiveJavaControl" tagprefix="elj" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Inline Editing Example</title>
</head>
<body>
  <form id="form1" runat="server" method="post">
    <elj:EditLiveJava ID="EditLiveJava1" runat="server"
      DownloadDirectory="../../redistributables/editlivejava" />
  </form>
</body>
</html>
```

2. Add the [Inline Editing](#) attribute to the **EditLiveJava** tag, specifying *true*.

```
...
  <elj:EditLiveJava ID="EditLiveJava1" runat="server" InlineEditing="true"
    DownloadDirectory="../../redistributables/editlivejava" />
...
```

3. Create an **<elj:EditableSection>** tag for each [Inline Editing](#) section, specifying an ID. Populate the Content attribute with the content you'd like loaded in this [Inline Editing](#) section.

Previously, a separate **<div>** tag was required for each **<elj:EditableSection>** tag. The separate **<div>** is no longer needed.

```
...
  <elj:editlivejava id="EditLiveJava1" runat="server" InlineEditing="True"
    DownloadDirectory="../../redistributables/editlivejava /><br />
```

```

<elj:EditableSection ID="EditableSection1" Content="&lt;p&gt;content 1&lt;/p&gt;"
  runat="server" Height="200px" /><br />
<elj:EditableSection ID="EditableSection2" Content="&lt;p&gt;content 2&lt;/p&gt;"
  runat="server" Height="200px" /><br />
...

```

## Retrieving Content from EditLive! in InlineEditingExample.aspx

Extracting content from EditLive! [Inline Editing](#) sections is the same as for standard ASP.NET controls. You simply retrieve the value of the Content attribute on the EditableSection control.

In our example, the event-handler for a button populates two server-side <div> tags with the content of our two [Inline Editing](#) sections.

**aspx page:**

```

...
<form id="form1" runat="server" method="post">
  <elj:EditLiveJava ID="EditLiveJava1" runat="server" InlineEditing="true"
    DownloadDirectory="../../redistributables/editlivejava" />
  <elj:EditableSection ID="EditableSection1" Content="&lt;p&gt;content 1&lt;/p&gt;"
    runat="server" Height="200px" /><br />
  <elj:EditableSection ID="EditableSection2" Content="&lt;p&gt;content 2&lt;/p&gt;"
    runat="server" Height="200px" /><br />

  <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" /><br />

  <asp:TextBox ID="TextBox1" runat="server" Height="91px" TextMode="MultiLine"
    Width="256px" /><br />
  <asp:TextBox ID="TextBox2" runat="server" Height="88px" TextMode="MultiLine"
    Width="255px" />

  </form>
...

```

**c# codebehind:**

```

using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class inlineEditing_InlineEditingExample : System.Web.UI.Page {
    protected void Button1_Click(object sender, EventArgs e) {
        TextBox1.Text = EditableSection1.Content;
        TextBox2.Text = EditableSection2.Content;
    }
}

```

By default ASP.NET forms cannot post values containing HTML. In order to successfully post EditLive!'s content, you will need to ensure the page directive contains the **validateRequest="false"** parameter.

In ASP.NET 4.0, you also need to add the following to your web.config, in the <system.web> section:

```
<httpRuntime requestValidationMode="2.0" />
```



# Inline Editing in ASP.NET Example Code

## aspx page:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="InlineEditingExample.aspx.cs" Inherits="
inlineEditing_InlineEditingExample" ValidateRequest="false" %>
<%@ Register assembly="EditLiveJavaControl" namespace="EditLiveJavaControl" tagprefix="elj" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Inline Editing Example</title>
</head>
<body>
    <form id="form1" runat="server" method="post">
        <elj:EditLiveJava ID="EditLiveJava1" runat="server" InlineEditing="true" DownloadDirectory="../..
/redistributables/editlivejava" />
        <elj:EditableSection ID="EditableSection1" Content="&lt;p&gt;content 1&lt;/p&gt;" runat="server"
Height="200px" /><br />
        <elj:EditableSection ID="EditableSection2" Content="&lt;p&gt;content 2&lt;/p&gt;" runat="server"
Height="200px" /><br />

        <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" /><br />

        <asp:TextBox ID="TextBox1" runat="server" Height="91px" TextMode="MultiLine" Width="256px" /><br />
        <asp:TextBox ID="TextBox2" runat="server" Height="88px" TextMode="MultiLine" Width="255px" />

    </form>
</body>
</html>
```

## c# codebehind:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class inlineEditing_InlineEditingExample : System.Web.UI.Page {
    protected void Button1_Click(object sender, EventArgs e) {
        TextBox1.Text = EditableSection1.Content;
        TextBox2.Text = EditableSection2.Content;
    }
}
```

# Basic ColdFusion Example

## Documentation

The [Basic ColdFusion Example Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code view for all the associated files in the Basic ColdFusion example is available [here](#).

# Basic ColdFusion Example Documentation

This section of the document provides information on how to integrate EditLive! into a Web page using ColdFusion and JavaScript. The complete source code for this example can be found in the *EDITLIVE\_INSTALL/webfolder/coldfusion/basic/* folder, where *EDITLIVE\_INSTALL* is the location that the EditLive! SDK has been installed.

## Getting Started

### Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic ColdFusion

## Overview

In this sample EditLive! is embedded into a Web page using ColdFusion and JavaScript.

This example demonstrates how to perform the following with EditLive! and ColdFusion:

- Embed an instance of EditLive! in a Web page using JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive!.
- Load a document into EditLive! from a ColdFusion variable.

## Integrating EditLive!

To embed EditLive! within a Web page several steps are required. Each of these steps is explained here and code samples are provided.

1. Include the *editlivejava.js* file

```
<script src="../../redistributables/editlivejava/editlivejava.js"> </script>
```

The *editlivejava.js* file contains the Tiny EditLive! JavaScript library. This library provides the interface between the browser and the EditLive! .jar file (*editlivejava.jar*) which contains the code for the EditLive! applet. The JavaScript library file can be found in the *EDITLIVE\_INSTALL/redistributables/editlivejava* directory, where *EDITLIVE\_INSTALL* is where the EditLive! SDK was installed.

2. Declare the EditLive! JavaScript object.

```
<script language="JavaScript"> var editliveInstance;
```

3. Create a new instance of the EditLive! object. When creating the EditLive! object, the name of the form field for the applet, in addition to the width and height, are declared. In this example, the form field for the applet is *ELApplet1*, the width of the applet is 700 pixels, and the height is 600 pixels.

```
// Create a new EditLive! instance with the name  
// "ELApplet1", a height of 600 pixels and a width of 700 pixels.  
editliveInstance = new EditLiveJava("ELApplet1", 700, 600);
```

4. Set the path to the source files for EditLive!. These can be found in the *EDITLIVE\_INSTALL/webfolder/redistributables/editlivejava* directory.

```
// This sets a relative path to the directory where  
// the EditLive! redistributables can be  
// found e.g. editlivejava.jar  
editliveInstance.setDownloadDirectory( "../../redistributables/editlivejava");
```

5. Load the EditLive! configuration file into a string and set the configuration text.

```
<!-- Load the configuration file on the server to speed up loading --> <cfset configpath=ExpandPath  
("sample_eljconfig.xml")> <cffile action="read" file="#configpath#" variable="xmlConfig"> <cfoutput>  
editliveInstance.setConfigurationText("#URLEncodedFormat(xmlConfig)#"); </cfoutput>
```

`URL`EncodedFormat must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

6. Set the content for the applet via the [setBody Method](#) (the [setDocument Method](#) may also be used). The content must be URL encoded. The following example will first set the variable contents to "*<p>Document contents of EditLive!</p>*" and then set the content of EditLive! to the contents of the variable contents.

```
<cfset contents="<p>This is the initial source</p>">
<cfoutput>
    editliveInstance.setBody( "#URLEncodedFormat( contents)#" );
</cfoutput>
```

The contents variable can contain any HTML fragment - for example, content loaded from a database. Note that URLEncodedFormat must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

7. Display the EditLive! applet and close the script element.

```
// .show is the final call and instructs the JavaScript
// library (editlivejava.js) to insert a new EditLive!
// at the this location.
editliveInstance.show(); </script>
```

This section of code creates an instance of EditLive! within the page and sets properties which affect how EditLive! will be presented within the page. For more information on each of the load-time properties here (the [JavaScript Constructor](#), [setConfigurationFile Method](#), [setBody Method](#), and [show Method](#)), see the EditLive! [Load Time Methods](#) article. After each of the properties have been set the show property is invoked. This property causes the instance of EditLive! to be displayed in the Web page.

# Basic ColdFusion Example Code

```
<!-- Load the configuration file on the server to speed up loading --->
<cfset configpath=ExpandPath("config.xml")>
<cffile action="read" file="#configpath#" variable="xmlConfig">

<cfset contents="<p>This is the initial source</p>">

<html>
  <head>
    <title>Sample EditLive! ColdFusion Integration</title>
    <script src="../../redistributables/editlivejava/editlivejava.js">
    </script>
  </head>
  <body>
    <cfoutput>
      <script language="JavaScript">
        var editliveInstance;
        // Create a new EditLive! instance with the name
        // "ELApplet1", a height of 600 pixels and a width of 700 pixels.
        editliveInstance = new EditLiveJava("ELApplet1", 700, 600);
        // This sets a relative path to the directory where
        // the EditLive! redistributables can be
        // found e.g. editlivejava.jar
        editliveInstance.setDownloadDirectory(
          "../../redistributables/editlivejava"
        );
        // This sets a relative or absolute path to the XML
        // configuration file to use.
        editliveInstance.setConfigurationText("#URLEncodedFormat(xmlConfig)#");
        // This sets the initial content to be displayed within
        // EditLive!
        editliveInstance.setBody("#URLEncodedFormat(contents)#");
        // .show is the final call and instructs the JavaScript
        // library (editlivejava.js) to insert a new EditLive!
        // at the this location.
        editliveInstance.show();
      </script>
    </cfoutput>
  </body>
</html>
```

# Basic PHP Example

## Documentation

The [Basic PHP Example Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code view for all the associated files in the Basic PHP example is available [here](#).

# Basic PHP Example Documentation

This article provides information on how to integrate EditLive! into a Web page using PHP and JavaScript. The complete source code for this example can be found in the *EDITLIVE\_INSTALL/webfolder/php/basic/* directory, where *EDITLIVE\_INSTALL* is the location that the EditLive! SDK has been installed.

## Getting Started

### Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic PHP

### Overview

In this sample EditLive! is embedded into a Web page using PHP and JavaScript.

This example demonstrates how to perform the following with EditLive! and PHP:

- Embed an instance of EditLive! in a Web page using JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive!.
- Load a document into EditLive! from a PHP variable.

## Integrating EditLive!

To embed EditLive! within a Web page several steps are required. Each of these steps is explained here and code samples are provided.

1. Include the editlivejava.js file

```
<script src="../../redistributables/editlivejava/editlivejava.js">
```

The editlivejava.js file contains the Ephox EditLive! JavaScript library. This library provides the interface between the browser and the EditLive! .jar file (editlivejava.jar) which contains the code for the EditLive! applet. The JavaScript library file can be found in the *EDITLIVE\_INSTALL/redistributables/editlivejava* directory of the EditLive! install.

2. Declare the EditLive! JavaScript object.

```
<script language="JavaScript"> var editliveInstance;
```

3. Create a new instance of the EditLive! object. When creating the EditLive! object, the name of the form field for the applet, in addition to the width and height, are declared. In this example, the form field for the applet is ELApplet1, the width of the applet is 700 pixels, and the height is 600 pixels.

```
// Create a new EditLive! instance with the name  
// "ELApplet1", a height of 600 pixels and a width of 700 pixels.  
editliveInstance = new EditLiveJava("ELApplet1", 700, 600);
```

4. Set the path to the source files for EditLive!. These can be found in the *INSTALL\_HOME/webfolder/redistributables/editlivejava* directory.

```
// This sets a relative path to the directory where  
// the EditLive! redistributables can be  
// found e.g. editlivejava.jar  
editliveInstance.setDownloadDirectory( "../../redistributables/editlivejava");
```

5. Load the EditLive! configuration file into a string and set the configuration text.

```
<?  
//load the XML file into the string "$xmlConfig"  
// this helps to speed up the ELJ load time  
  
$filename = "sample_eljconfig.xml";  
$fd = fopen($filename,"r");
```

```

$xmlConfig = fread ($fd, filesize($filename));
fclose ($fd);
?>

editliveInstance.setConfigurationText("<?rawurlencode($xmlConfig)?>");

```

Note: the PHP function `rawurlencode` must be used to output the variable. For more information, see the article on [Encoding Content for Use with EditLive!](#).

6. Set the content for the applet via the [setBody Method](#) (the [setDocument Method](#) may also be used). The content must be URL encoded. The following example will first set the variable `$pageContent` to "`<p>Document contents of EditLive!</p>`" and then set the content of EditLive! to the contents of the variable `$pageContent`.

```

// This sets the initial content to be displayed within
// EditLive!
<?php
    $pageContent = "<p>Document contents of EditLive!</p>"
?>

editliveInstance.setBody("<?rawurlencode($pageContent)?>");

```

The `$pageContent` variable can contain any HTML fragment, for example content loaded from a database. Note that `rawurlencode` must be used to output the variable. For more information see the article on [Encoding Content for Use with EditLive!](#).

7. Display the EditLive! applet and close the script element.

```

// .show is the final call and instructs the JavaScript
// library (editlivejava.js) to insert a new EditLive!
// at the this location.
editliveInstance.show();
</script>

```

This section of code creates an instance of EditLive! within the page and sets properties which affect how EditLive! will be presented within the page. For more information on each of these load-time properties (the JavaScript constructor, [setConfigurationFile](#), [setBody](#), and [show](#)), see the EditLive! [Load Time Methods](#). After each of the properties have been set the [show Method](#) is called. This method causes the instance of EditLive! to be displayed in the Web page.



# Basic PHP Code

```
<?php
    //load the XML file into the string "$xmlConfig"
    // this helps to speed up the ELJ load time
    $filename = "config.xml";
    $fd = fopen($filename,"r");
    $xmlConfig = fread ($fd, filesize($filename));
    fclose ($fd);

    $pageContent = "<p>Document contents of EditLive!</p>"
?>
<html>
  <head>
    <title>Sample EditLive! PHP Integration</title>
    <script src="../../redistributables/editlivejava/editlivejava.js">
    </script>
  </head>
  <body>
    <script language="JavaScript">
      var editliveInstance;
      // Create a new EditLive! instance with the name
      // "ELApplet1", a height of 600 pixels and a width of 700 pixels.
      editliveInstance = new EditLiveJava("ELApplet1", 700, 600);
      // This sets a relative path to the directory where
      // the EditLive! redistributables can be
      // found e.g. editlivejava.jar
      editliveInstance.setDownloadDirectory(
        "../../redistributables/editlivejava"
      );
      // This sets a relative or absolute path to the XML
      // configuration file to use.
      editliveInstance.setConfigurationText("<?rawurlencode($xmlConfig)?>");
      // This sets the initial content to be displayed within
      // EditLive!
      editliveInstance.setBody("<?rawurlencode($pageContent)?>");
      // .show is the final call and instructs the JavaScript
      // library (editlivejava.js) to insert a new EditLive!
      // at the this location.
      editliveInstance.show();
    </script>
  </body>
</html>
```

# ColdFusion in MySQL Example

## Documentation

The [MySQL in ColdFusion Example Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code views for all the associated files in the Basic ASP example are available [here](#).

# MySQL in ColdFusion Example Documentation

This article shows how to use EditLive! as the interface for a database driven press release center. This sample allows users to:

- add new press articles,
- edit existing press articles,
- view the article in the web browser, and
- delete existing press articles.

The complete source code for this example can be found in the *EDITLIVE\_INSTALL/webfolder/examples/coldfusion/database* folder, where *EDITLIVE\_INSTALL* is the location that the EditLive! for Java SDK has been installed to.

## Getting Started

### Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic ColdFusion
- Basic MySQL

## Overview

In this sample EditLive! is embedded into a Web page using ColdFusion and JavaScript. The example sets several variables affecting the appearance and functionality of the applet and loads a basic document into the instance of the applet.

This example demonstrates how to perform the following with EditLive! and ColdFusion:

- Embed an instance of EditLive! in a Web page using JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive!.
- Load a document into EditLive! from a ColdFusion variable.

## Creating the Database

The database used in this sample is a MySQL database named ELContent. Below is a table outlining the fields within the database table articles and a description of the information each field stores.

Column Name	Description
article_id	A unique number used to identify the article record
article_title	The title or headline of the article
article_body	The actual article content
article_styleElementText	The styles used to format the article if content was pasted in to EditLive! for Microsoft Word

To create this table, use the following MySQL query:

```
CREATE TABLE `articles` (  
  `article_id` int(11) NOT NULL auto_increment,  
  `article_title` varchar(255) collate utf8_unicode_ci NOT NULL default '',  
  `article_styleElementText` blob NOT NULL,  
  `article_body` blob NOT NULL,  
  PRIMARY KEY (`article_id`)  
) DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

To create the entire database including sample content, import the database.sql file. Use the following command, which will prompt you for the mysql root password:

```
mysql -u root -p < database.sql
```

## Creating the ColdFusion Data Source

This example uses a ColdFusion data source called ELContent. To add a data source:

1. Start the ColdFusion Administrator. For ColdFusion MX 7 on Windows, Select **Start -> Programs -> Macromedia -> ColdFusion MX 7 -> Administrator**.
2. Enter the **Administrator** password if prompted.
3. Choose **Datasources** from the **Data & Services** section on the left.
4. Enter the Data Source Name **ELContent**.
5. Choose the Driver **MySQL**.
6. Click **Add**.
7. Enter the database name **ELContent**.
8. Enter the server, username and password for your MySQL server.
9. Click **Submit**.
10. Locate ELContent in the list and click **Verify**. This will run a test to ensure the data source connection is correct. You will see a message stating that the connection to the data source was verified correctly.

## Integrating EditLive!

To use EditLive! with a database, several web pages are required. Each of these pages is explained here and code samples are provided.

### Index Page (start.cfm)

The index page of the press release center lists all of the articles currently available in the database. Users are able to:

- create a new article using EditLive!,
- edit an existing article using EditLive!, and
- delete an existing article from the press database.

1. Create a link in the page to the file *add.cfm* to allow users to use EditLive! to create a new article.

```
<P><A href="add.cfm">Create a new article</A></P>
```

2. Connect to the database and retrieve all of the existing articles listed in the articles table.

```
<!--Create a query to retrieve a list of articles from the database-->
<CFQUERY DATASOURCE="ELContent" NAME="article_list">
  SELECT * FROM articles ORDER BY article_id
</CFQUERY>

<!--Check if there are records in the database.-->
<CFIF article_list.RecordCount IS "0">
```

3. Let users know if there are no records in the database.

```
<!--There are no records. Tell the user.-->
<p>There are no records in the database. Click <STRONG>Add an Article</STRONG> to add a record to the
  database.</p>
```

4. Use embedded ColdFusion to loop through the recordset of articles to create a table which lists all of the existing articles by title. Then, for each article, create three links:

- View page to view the article
- Edit page to edit the article using EditLive!
- Delete page to delete the article from the database

Append the *article\_id* to all of the links so that the relevant article is known.

```
<CFELSE>
  <!-- There are records in the database. Loop through all the records in the query and
    write out a table row for each record. Include links to view the article, edit the
    article, or delete the article.
  -->

  <TABLE cellpadding=3 cellspacing=0 border=0>
    <TR>
      <TH>ID</TH>
```

```

        <TH width="250">Title</TH>
        <TH>Available Actions</TH>
    </TR>

    <CFOUTPUT query="article_list">
        <TR>
            <TD>#article_id#</TD>
            <TD>#article_title#</TD>
            <TD><A href="view.cfm?article_id=#article_id#">View</A> |
                <A href="edit.cfm?article_id=#article_id#">Edit</A> |
                <A href="delete.cfm?article_id=#article_id#">Delete</A>
            </TD>
        </TR>
    </CFOUTPUT>
</TABLE>
</CFIF>

```

### Adding a New Article (add.cfm and xt\_add.cfm)

The file *add.cfm* is used to create a new article using EditLive!. Once the article is complete, it is then added to the database using the processing page, *xt\_add.cfm*.

#### add.cfm

1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see EditLive! Basic ColdFusion Integration.
2. Add a heading to the page of "Create a New Article".

```
<BODY> <H1>Create a New Article</H1>
```

3. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt\_add.cfm*, in the action attribute of the FORM tag.

```

<!-- This form contains EditLive!, a text area for the article title, a submit button and a cancel button.
-->
<FORM action="xt_add.cfm" method="POST" name="articleForm">

```

4. Add text field at the beginning of the form for the article title.

```

<!-- Article title -->
<P>Title: <INPUT type="text" name="article_title" size="50"></P>

```

5. Create a string with the initial content for the new page.

```
<cfset contents="<p>This is the initial source</p>">
```

6. Modify the setBody() call to use the new content variable.

```
ELJApplet1.js.setBody("#URLEncodedFormat(contents)#");
```

URLEncodedFormat must be used to output the variable; for more information, see the article on [Encoding Content for Use with EditLive!](#).

7. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```

<P>
    <INPUT type="submit" value="Save" name="Add Article">
    <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>

```

8. Close the form tag.

```
</form>
```

### xt\_add.cfm

1. Generate the SQL query to insert the new content into the database, ensuring all content is escaped for SQL.

```
<!--  
Update the article record values in the database with the values from the form objects. Use <cfqueryparam> to  
escape the content for SQL.  
-->  
<CFQUERY datasource="ELContent" name="add_article">  
  INSERT INTO articles (article_title, article_body, article_styleElementText)  
  VALUES(<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,  
         <cfqueryparam value="#FORM.ELJApplet1#" cfsqltype="cf_sql_varchar">,  
         <cfqueryparam value="#FORM.ELJApplet1_styles#" cfsqltype="cf_sql_varchar">)  
</CFQUERY>
```

2. Redirect to start.cfm.

```
<!-- Go back to the start page --> <CFLOCATION URL="start.cfm">
```

### Editing an Existing Article (edit.cfm and xt\_edit.cfm)

The file *edit.cfm* loads an existing article into EditLive! for modification. Once the article is complete, the relevant record in the database is updated using the processing page, *xt\_edit.cfm*.

#### edit.cfm

1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see EditLive! [Basic ColdFusion Example](#).
2. Use <cfparam> to require that the **article\_id** parameter has been specified.

```
<!-- throw an error if article_id was not specified -->  
<cfparam name="article_id">
```

3. Add a heading to the page of "Edit Document".

```
<BODY> <H1>Edit Document</H1>
```

4. Retrieve the current article. Ensure the articleID variable is escaped for SQL.

```
<!--  
Retrieve the specified article details from the database Use <cfqueryparam> to escape the content for SQL.  
-->  
<CFQUERY DATASOURCE="ELContent" NAME="article_content">  
  SELECT * FROM articles WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">  
</CFQUERY>
```

5. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt\_edit.cfm*, in the action attribute of the FORM tag.

```
<!--  
This form contains EditLive!, a text area for the article title, a submit button and a cancel button.  
-->  
<FORM action="xt_add.cfm" method="POST" name="articleForm">
```

6. Create a hidden form object for the article ID.

```
<cfoutput>  
  <!-- Hidden field for identifying the article -->  
  <INPUT type="hidden" name="article_id" value="#article_content.article_id#">
```

7. Add a text field at the beginning of the form for the article title.

```
<!-- Article title -->  
<P>Title:
```

```
<INPUT type="text" name="article_title" value="#HTMLEditFormat(article_content.article_title)#" size="40">
</P>
```

8. Modify the `setBody()` call to load the content from the database.

```
ELJApplet1_js.setBody("#URLEncodedFormat(article_content.article_body)#");
```

`URLEncodedFormat` must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

9. Add a `setStyles()` call directly after `setBody()` to load the styles from the database.

```
ELJApplet1_js .setStyles("#URLEncodedFormat(article_content.article_styleElementText)#");
```

`URLEncodedFormat` must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

10. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```
<P>
  <INPUT type="submit" value="Save" name="Add Article">
  <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>
```

11. Close the form tag.

```
</form>
```

#### **xt\_edit.cfm**

1. Generate the SQL query to update the article in the database, ensuring all content is escaped for SQL.

```
<!---
Update the article record values in the database with the values from the form objects. Use <cfqueryparam> to
escape the content for SQL.
--->

<CFQUERY DATASOURCE="ELContent">
UPDATE articles
SET article_title=<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,
    article_body=<cfqueryparam value="#FORM.article_body#" cfsqltype="cf_sql_varchar">,
    article_styleElementText=<cfqueryparam value="#FORM.article_styleElementText#" cfsqltype="cf_sql_varchar">
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

2. Redirect to `start.cfm`.

```
<!--- Go back to the start page --->
<CFLOCATION URL="start.cfm">
```

#### Deleting an Article (`delete.cfm` and `xt_delete.cfm`)

The file `delete.cfm` displays the chosen article information to allow the user to confirm that they wish to delete the specified article. Once confirmation is obtained, the relevant record in the database is deleted using the processing page, `xt_delete.cfm`.

#### **delete.cfm**

1. Use `<cfparam>` to require that the `article_id` parameter has been specified.

```
<!--- throw an error if article_id was not specified --->
<cfparam name="article_id">
```

2. Retrieve the current article information. Ensure the `articleID` variable is escaped for SQL.

```

<!-- Retrieve the specified article details from the database --->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles WHERE article_id=<cfqueryparam value="#article_id#"
      cfsqltype="cf_sql_integer">
</CFQUERY>

```

3. Create a standard HTML page, adding the heading "Delete a Document" and a message asking the user if they wish to delete the article.

```

<HTML>
  <HEAD>
    <CFOUTPUT>
      <TITLE>Delete Article ID #article_content.article_id#</TITLE>
    </CFOUTPUT>
  </HEAD>
  <BODY>
    <H1>Delete a Document</H1>
    <P>Are you sure you wish to delete this article?</P>

```

4. Create a table in the HTML page that displays the article id and title using the variable defined in Step 2.

```

<!--
This table contains the article ID and title for deletion confirmation.
-->

<TABLE>
<CFOUTPUT>
  <!-- Article ID -->
  <TR>
    <TD>Article ID:</TD>
    <TD>#article_content.article_id#</TD>
  </TR>

  <!-- Article title -->
  <TR>
    <TD>Title:</TD>
    <TD>#article_content.article_title#</TD>
  </TR>
</TABLE>

```

5. Create a form in the page. Put the *xt\_delete.cfm* file name in the action attribute of the FORM tag.

```

<!-- Form for deleting the article --->
<FORM action="xt_delete.cfm" method="POST">

```

6. Add a hidden form object to the form to store the article id for processing.

```

<INPUT type="hidden" name="article_id" value="#article_content.article_id#">

```

7. Add buttons to delete the article or cancel the deletion.

```

<P>
  <INPUT type="submit" value="Delete">
  <INPUT type="button" value="Cancel" onclick="javascript:history.back();">
</P>

```

8. Close the form and document tags.

```

  </FORM>
</CFOUTPUT>
</BODY>
</HTML>

```



## xt\_delete.cfm

1. Generate the SQL query to remove the article from the database, ensuring the article id is escaped for SQL.

```
<!-- Delete the specified article from the database -->
<CFQUERY DATASOURCE="ELContent" name="article_delete">
DELETE FROM articles
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

2. Redirect to start.cfm

```
<!-- Go back to the start page -->
<CFLOCATION URL="start.cfm">
```

## Viewing an Article (view.cfm)

The file view.cfm displays the chosen article.

1. Use <cfparam> to require that the **article\_id** parameter has been specified.

```
<!-- throw an error if article_id was not specified -->
<cfparam name="article_id">
```

2. Retrieve the current article. Ensure the articleID variable is escaped for SQL.

```
<!-- Retrieve the specified article details from the database -->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles
WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

3. Use embedded ColdFusion to set the <STYLE> tag of the page to the article style information retrieved from the database and to display the article title and content.

```
<HTML>
  <HEAD>
    <!-- Display the article content -->
    <CFOUTPUT>
      <TITLE>#article_content.article_title#</TITLE>
      <STYLE> #article_content.article_styleElementText#</STYLE>
    </HEAD>
    <BODY>
      <H1>#article_content.article_title#</H1>
      #article_content.article_body#
    </CFOUTPUT>
  </BODY>
</HTML>
```

## Handling Image Uploads

To enable the image upload functionality, a new script must be created and then referenced in the configuration file.

To create and reference an upload script, refer to the [ColdFusion HTTP File Upload Handler Script](#) section in the EditLive! for Java SDK.

# MySQL in ColdFusion Code

- [add.cfm](#)
- [delete.cfm](#)
- [edit.cfm](#)
- [start.cfm](#)
- [view.cfm](#)
- [xt\\_add.cfm](#)
- [xt\\_delete.cfm](#)
- [xt\\_edit.cfm](#)

# add.cfm

```
<!---
*****
add.cfm -- create a new article

End users can enter content using a web form and EditLive!
The web form is submitted to the page xt_add.asp

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->

<!--- Load the configuration file on the server to speed up loading --->
<cfset configpath=ExpandPath("db_config.xml")>
<cffile action="read" file="#configpath#" variable="xmlConfig">

<cfset contents="<p>This is the initial source</p>">

<HTML>

<HEAD>
<TITLE>Create a New Article</TITLE>
<LINK rel="stylesheet" href="sample.css">

<!---
These script files initialize the EditLive! API so that the EditLive!
properties and methods may be set to customise EditLive!.
-->
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

</HEAD>

<BODY>

<H1>Create a New Article</H1>

<!---
This form contains EditLive!, a text area for the article title,
a submit button and a cancel button.
-->
<FORM action="xt_add.cfm" method="POST" name="articleForm">

  <!--- Article title --->
  <P>Title: <INPUT type="text" name="article_title" size="50"></P>
  <P>Body:<BR>

<cfoutput>
<script language="JavaScript">
var ELJApplet1_js;
ELJApplet1_js = new EditLiveJava("ELJApplet1", "700", "600");

ELJApplet1_js.setDownloadDirectory("../redistributables/editlivejava");
ELJApplet1_js.setConfigurationText("#URLEncodedFormat(xmlConfig)#");
ELJApplet1_js.setBody("#URLEncodedFormat(contents)#");

ELJApplet1_js.setLocalDeployment(false);
ELJApplet1_js.setAutoSubmit(true);
ELJApplet1_js.setDebugLevel("info");
ELJApplet1_js.setShowSystemRequirementsError(true);
ELJApplet1_js.show();
</script>
</cfoutput>
</P>
```

```
<P><INPUT type="submit" value="Save" name="Add Article"> <INPUT type="button" value="Cancel" name="Cancel"
onclick="javascript:history.back();"></P>
</FORM>

</BODY>

</HTML>
```

# delete.cfm

```
<!---
*****
delete.cfm -- confirm the user wants to delete an article

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
-->

<!-- throw an error if article_id was not specified -->
<cfparam name="article_id">

<!-- Retrieve the specified article details from the database -->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles
WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>

<HTML>
<HEAD>

<CFOUTPUT>
<TITLE>Delete Article ID #article_content.article_id#</TITLE>
</CFOUTPUT>
<LINK rel="stylesheet" href="sample.css">

</HEAD>

<BODY>

<H1>Delete a Document</H1>
<P>Are you sure you wish to delete this article?</P>

<!--
This table contains the article ID and article title for deletion confirmation.
-->

<TABLE>
<CFOUTPUT>
  <!-- Article ID -->
  <TR>
    <TD>Article ID:</TD>
    <TD>#article_content.article_id#</TD>
  </TR>

  <!-- Article title -->
  <TR>
    <TD>Title:</TD>
    <TD>#article_content.article_title#</TD>
  </TR>
</TABLE>

<!-- Form for deleting the article -->
<FORM action="xt_delete.cfm" method="POST">
  <INPUT type="hidden" name="article_id" value="#article_content.article_id#">
  <P><INPUT type="submit" value="Delete"> <INPUT type="button" value="Cancel" onclick="javascript:history.
back();"></P>
</FORM>
</CFOUTPUT>

</BODY>
```

</HTML>

# edit.cfm

```
<!---
*****
edit.cfm -- update an existing article

End users can enter content using a web form and EditLive!
The web form is submitted to the page xt_edit.asp

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
--->

<!--- throw an error if article_id was not specified --->
<cfparam name="article_id">

<!--- Load the configuration file on the server to speed up loading --->
<cfset configpath=ExpandPath("db_config.xml")>
<cffile action="read" file="#configpath#" variable="xmlConfig">

<!---
Retrieve the specified article details from the database
Use <cfqueryparam> to escape the content for SQL.
--->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles
WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>

<HEAD>
<CFOUTPUT>
<TITLE>Editing Article ID #article_content.article_id#</TITLE>
</CFOUTPUT>
<LINK rel="stylesheet" href="sample.css">

<!---
These script files initialize the EditLive! API so that the EditLive!
properties and methods may be set to customise EditLive!.
--->
<script src="../../redistributables/editlivejava/editlivejava.js"></script>

</HEAD>

<BODY>

<H1>Edit Document</H1>

<!---
This form contains EditLive!, a text area for the article title,
a submit button and a cancel button.
--->
<FORM action="xt_edit.cfm" method="POST" name="articleForm">

<cfoutput>
  <!--- Hidden field for identifying the article --->
  <INPUT type="hidden" name="article_id" value="#article_content.article_id#">

  <!--- Article title --->
  <P>Title: <INPUT type="text" name="article_title" value="#HTMLFormat(article_content.article_title)#"
size="40"></P>
  <P>Body: <BR>

<script language="JavaScript">
```

```
var ELJApplet1_js;
ELJApplet1_js = new EditLiveJava("ELJApplet1", "700", "600");

ELJApplet1_js.setDownloadDirectory("../..//redistributables/editlivejava");
ELJApplet1_js.setConfigurationText("#URLEncodedFormat(xmlConfig)#");
ELJApplet1_js.setBody("#URLEncodedFormat(toString(article_content.article_body))#");
ELJApplet1_js.setStyles("#URLEncodedFormat(toString(article_content.article_body))#");

ELJApplet1_js.setLocalDeployment(false);
ELJApplet1_js.setAutoSubmit(true);
ELJApplet1_js.setDebugLevel("info");
ELJApplet1_js.setShowSystemRequirementsError(true);
ELJApplet1_js.show();
</script>
</cfoutput>

<P><INPUT type="submit" value="Save"> <INPUT type="button" value="Cancel" onclick="javascript:history.back();"
></P>
</FORM>

</BODY>

</HTML>
```



# start.cfm

```
<!---
*****
start.cfm -- the home page for the application

Lists the articles stored in the database and provides
links for creating a new article, editing existing articles
and deleting existing articles

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
--->
<HTML>

<HEAD>
<TITLE>Sample Database Application for Coldfusion</TITLE>
<LINK rel="stylesheet" href="sample.css">
</HEAD>

<BODY>

<H1>Sample Database Application for Coldfusion</H1>

<P><A href="add.cfm">Create a new article</A></P>

<!--Create a query to retrieve a list of articles from the database-->
<CFQUERY DATASOURCE="ELContent" NAME="article_list">
SELECT * FROM articles ORDER BY article_id
</CFQUERY>

<!--Check if there are records in the database.-->
<CFIF article_list.RecordCount IS "0">

<!--There are no records. Tell the user.-->

<p>There are no records in the database. Click <STRONG>Add an Article</STRONG>
to add a record to the database.</p>

<CFELSE>
<!--
There are records in the database. Loop
through all the records in the query
and write out a table row for each record.
Include links to view the article, edit the
article, or delete the article.
--->

<TABLE cellpadding=3 cellspacing=0 border=0>
  <TR>
    <TH>ID</TH>
    <TH width="250">Title</TH>
    <TH>Available Actions</TH>
  </TR>

<CFOUTPUT query="article_list">
  <TR>
    <TD>#article_id#</TD>
    <TD>#article_title#</TD>
    <TD><A href="view.cfm?article_id=#article_id#">View</A> |
    <A href="edit.cfm?article_id=#article_id#">Edit</A> |
    <A href="delete.cfm?article_id=#article_id#">Delete</A></TD>
  </TR>
```

```
</CFOUTPUT>  
</TABLE>  
  
</CFIF>  
  
</BODY>  
</HTML>
```

# view.cfm

```
<!---
*****
view.asp -- template for displaying an article

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****
--->

<!-- throw an error if article_id was not specified --->
<cfparam name="article_id">

<!-- Retrieve the specified article details from the database --->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles
WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>

<HTML>
<HEAD>

<!-- Display the article content --->
<CFOUTPUT>
<TITLE>#article_content.article_title#</TITLE>
<STYLE>
#toString(article_content.article_styleElementText)#
</STYLE>
</HEAD>
<BODY>

<H1>#article_content.article_title#</H1>

#toString(article_content.article_body)#
</CFOUTPUT>

</BODY>
</HTML>
```

## xt\_add.cfm

```
<!---
Update the article record values in the database with the values from the form objects.
Use <cfqueryparam> to escape the content for SQL.
--->
<CFQUERY datasource="ELContent" name="add_article">
    INSERT INTO articles (article_title, article_body, article_styleElementText)
    VALUES(<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,
           <cfqueryparam value="#FORM.ELJApplet1#" cfsqltype="cf_sql_varchar">,
           <cfqueryparam value="#FORM.ELJApplet1_styles#" cfsqltype="cf_sql_varchar">)
</CFQUERY>

<!--- Go back to the start page --->
<CFLOCATION URL="start.cfm">
```

## xt\_delete.cfm

```
<!-- Delete the specified article from the database -->
<CFQUERY DATASOURCE="ELContent" name="article_delete">
DELETE FROM articles
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>

<!-- Go back to the start page -->
<CFLOCATION URL="start.cfm">
```

## xt\_edit.cfm

```
<!---
Update the article record values in the database with the values from the form objects.
Use <cfqueryparam> to escape the content for SQL.
--->
<CFQUERY DATASOURCE="ELContent">
UPDATE articles
SET article_title=<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,
    article_body=<cfqueryparam value="#FORM.ELJApplet1#" cfsqltype="cf_sql_varchar">,
    article_styleElementText=<cfqueryparam value="#FORM.ELJApplet1_styles#" cfsqltype="cf_sql_varchar">
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>

<!--- Go back to the start page --->
<CFLOCATION URL="start.cfm">
```

# MySQL in PHP Example

## Documentation

The [MySQL in PHP Example Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code views for all the associated files in the PHP and MySQL example are available [here](#).

# MySQL in PHP Example Documentation

## Introduction

This article shows how to use EditLive! as the interface for a database driven press release center. This sample allows users to:

- add new press articles,
- edit existing press articles,
- view the article in the web browser, and
- delete existing press articles.

The complete source code for this example can be found in the `EDITLIVE_INSTALL/webfolder/examples/php/database` folder where `EDITLIVE_INSTALL` is the location where the EditLive! SDK has been installed.

## Getting Started

### Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic PHP
- Basic MySQL

## Overview

In this sample, EditLive! is embedded into a Web page using PHP and JavaScript. The example sets several variables affecting the appearance and functionality of the applet and loads a basic document into the instance of the applet.

This example demonstrates how to perform the following with EditLive! and PHP:

- Embed an instance of EditLive! in a Web page using PHP and JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive!.
- Load a document into EditLive! from a PHP variable.

## Creating the Database

The database used in this sample is a MySQL database named ELContent. Below is a table outlining the fields within the database table articles and a description of the information each field stores.

Column Name	Description
article_id	A unique number used to identify the article record
article_title	The title or headline of the article
article_body	The actual article content
article_styleElementText	The styles used to format the article if content was pasted in to EditLive! for Microsoft Word

To create this table, use the following MySQL query:

```
CREATE TABLE `articles` (  
  `article_id` int(11) NOT NULL auto_increment,  
  `article_title` varchar(255) collate utf8_unicode_ci NOT NULL default '',  
  `article_styleElementText` blob NOT NULL,  
  `article_body` blob NOT NULL,  
  PRIMARY KEY (`article_id`)  
) DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

To create the entire database including sample content, import the `database.sql` file. Use the following command, which will prompt you for the mysql root password:

```
mysql -u root -p < database.sql
```



## Integrating EditLive! for Java

To use EditLive! with a database, several web pages are required. Each of these pages is explained here and code samples are provided.

### Database Connection Page (i\_database.php)

This file defines the functions used to interact with the database in order to ensure that the other files are not database-specific. The main functions used are DBConnect(), DBQuery(), DBFetchRow() and DBResult(). It also contains the function escape\_string() to escape content strings for use in SQL.

### Index Page (start.php)

The index page of the press release center lists all of the articles currently available in the database. Users are able to:

- create a new article using EditLive!,
- edit an existing article using EditLive!, and
- delete an existing article from the press database.

1. Include i\_database.php to connect to the database.

```
<?
    require_once("./i_database.php");
?>
```

2. Create a link in the page to the file add.php to allow users to use EditLive! to create a new article.

```
<P><A href="add.php">Create a new article</A></P>
```

3. Connect to the database and retrieve all of the existing articles listed in the articles table.

```
<?
    //Attempt to establish a connection with the database
    if (!DBConnect()) {
        print DBError();
    }
    else {
        DBQuery("select * from articles");
        if (DBFetchRow()) {
?>
```

4. Use embedded PHP to loop through the recordset of articles to create a table which lists all of the existing articles by title. Then, for each article, create three links:

- View page to view the article.
- Edit page to edit the article using EditLive!.
- Delete page to delete the article from the database.

Append the article\_id to all of the links so that the relevant article is known.

```
<TABLE cellpadding=3 cellspacing=0 border=0>
  <TR>
    <TH>ID</TH>
    <TH width="250">Title</TH>
    <TH>Available Actions</TH>
  </TR>
  <?
    // There are records. Loop through all the records in the
    // recordset and write out a table row for each record
    do {
?>
  <TR>
    <TD><?=DBResult("article_id")?></TD>
    <TD><?=DBResult("article_title")?></TD>
    <TD><A href="view.php?article_id=<?=DBResult("article_id")?>">View</A> |
```

```

        <A href="edit.php?article_id=?=DBResult("article_id")?>">Edit</A> |
        <A href="delete.php?article_id=?=DBResult("article_id")?>">Delete </A>
    </TD>
</TR>
</TABLE>
<?
    } while(DBFetchRow())
?>
</TABLE>

```

5. Add an ELSE statement to the IF loop to let users know if there are no records in the database.

```

<?
    } else {
?>
<P>There are no records in the database. Click <STRONG>Create a new article</STRONG> to add a record to the
database.</P>
<?
    } }
?>

```

6. Disconnect from the database.

```

<?
    DBDisconnect();
?>

```

### Adding a New Article (add.php and xt\_add.php)

The file add.php is used to create a new article using EditLive!. Once the article is complete, it is then added to the database using the processing page, *xt\_add.php*.

#### add.php

1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see the EditLive! [Basic PHP Example](#).
2. Add a heading to the page of "Create a New Article".

```

<BODY>
    <H1>Create a New Article</H1>

```

3. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt\_add.php*, in the action attribute of the FORM tag.

```

<?
    // This form contains EditLive!, a text area for the article title,
    // a submit button and a cancel button.
?>
<FORM action="xt_add.php" method="POST" name="articleForm">

```

4. Add a text field at the beginning of the form for the article title.

```

<?
    //Article title
?>
<P>Title: <INPUT type="text" name="article_title" size="50"></P>

```

5. Create a string with the initial content for the new page.

```

<?
    //the initial content to load into ELJ
    $contents = "<p>This is the initial source</p>"
?>

```

6. Modify the setBody() call to use the new content variable.

`rawurlencode` must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

7. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```
<P>
  <INPUT type="submit" value="Save" name="Add Article">
  <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>
```

8. Close the form tag.

```
</form>
```

#### **xt\_add.php**

1. Include `_i_database.php` and initialize the redirect flag.

```
// include the database wrapper functions
require_once("../i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;
```

2. Attempt to connect to the database, and cancel the redirect if there is an error.

```
if (!DBConnect()) {
    print DBError();
    $redirect = false;
} else {
```

3. Create 3 strings to store the article title, content, and styles passed via POST, and escape them for SQL.

```
// Get the POSTed data from the form and escape it for SQL using the
// function in i_database
$articleTitle = escape_string($_HTTP_POST_VARS["article_title"]);
$articleBody = escape_string($_HTTP_POST_VARS["ELJApplet1"]);
$articleStyleElementText = escape_string($_HTTP_POST_VARS["ELJApplet1_styles"]);
```

4. Generate the SQL query to insert the new content into the database.

```
// Insert the POSTed data into the database
$query = "INSERT INTO articles ( article_title, article_styleElementText, "
. " article_body ) VALUES ( '$articleTitle', "
. "'$articleStyleElementText', '$articleBody' )";
```

5. Execute the SQL query, and cancel the redirect if there is an error.

```
DBQuery($query);
if (DBError() != ""){
    print DBError(); $redirect = false;
}
```

6. Disconnect from the database.

```
// Disconnect is required when updating DBDisconnect(); }
```

7. If there have been no errors and the redirect flag is still set, redirect to `start.php`.

```
// redirect if no errors if($redirect){ Header("Location: start.php"); } ?>
```

## Editing an Existing Article (edit.php and xt\_edit.php)

The file *edit.php* loads an existing article into EditLive! for modification. Once the article is complete, the relevant record in the database is updated using the processing page, *xt\_edit.php*.

### edit.php

1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page, please see the EditLive! [Basic PHP Example](#).

2. Include *i\_database.php*.

```
<? // include the database wrapper functions
    require_once("./i_database.php");
```

3. Read the HTTP GET variable *article\_id* to find which article the user has requested to edit.

```
// get the article id
    $articleID = $_HTTP_GET_VARS["article_id"];
```

4. Attempt to connect to the database; if there is an error print it and cancel the script.

```
// Attempt to establish a connection with the database
    if (!DBConnect()) {
        print DBError();
    }else {
    ?>
```

5. Add a heading to the page of "Edit Document".

```
<BODY> <H1>Edit Document</H1>
```

6. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the *articleID* variable is escaped for SQL.

```
<?
    //SQL query to get required document from the database
    DBQuery("SELECT * FROM articles WHERE article_id = "
        . escape_string($articleID) . ";"");
    if (DBFetchRow()) {
    ?>
```

7. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt\_edit.php*, in the action attribute of the FORM tag.

```
<?
    // This form contains EditLive!, a text area for the article title,
    // a submit button and a cancel button.
    ?>
<FORM action="xt_edit.php" method="POST" name="articleForm">
```

8. Create a hidden form object for the article id.

```
<?
    // Hidden field for identifying the article
    ?>
<INPUT type="hidden" name="article_id" value="<?=htmlspecialchars($article_id)?>">
```

9. Add a text field at the beginning of the form for the article title.

```
<?
    // Article title
    ?>
<P>Title: <INPUT type="text" name="article_title" value="<?=htmlspecialchars(DBResult("article_title"))?>"
size="40"></P>
```

10. Modify the the `setBody()` call to load the content from the database.

```
ELJApplet1_js.setBody("<?=rawurlencode(DBResult("article_body"))?>");
```

`rawurlencode` must be used to output the variable; for more information see the article on [Encoding Content for Use with EditLive!](#).

11. Add a `setStyles()` call directly after `setBody()` to load the styles from the database.

```
ELJApplet1_js.setStyles("<?=rawurlencode( DBResult("article_styleElementText"))?>");
```

12. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```
<P>
  <INPUT type="submit" value="Save" name="Add Article">
  <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>
```

13. Close the form tag.

```
</form>
```

14. If the article cannot be found, an error will need to be displayed.

```
<?
} else {
    //There are no records matching this article_id //TODO: Handle and display meaningful error
}
}
```

15. Disconnect from the database.

```
DBDisconnect(); ?> </BODY>
```

### **xt\_edit.php**

1. Include `_i_database.php` and initialize the redirect flag.

```
// include the database wrapper functions
require_once("../i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;
```

2. Attempt to connect to the database, and cancel the redirect if there is an error.

```
if (!DBConnect()) {
    print DBError();
    $redirect = false;
} else {
```

3. Create 4 strings to store the article title, content and styles passed via POST, and escape them for SQL.

```
// Get the POSTed data from the form and escape it for SQL using the
// function in i_database
$articleID = escape_string($HTTP_POST_VARS["article_id"]);
$articleTitle = escape_string($HTTP_POST_VARS["article_title"]);
$articleBody = escape_string($HTTP_POST_VARS["ELJApplet1"]);
$articleStyleElementText = escape_string($HTTP_POST_VARS["ELJApplet1_styles"]);
```

4. Generate the SQL query to insert the new content into the database.

```
// Insert the POSTed data into the database
$query = "UPDATE articles SET article_title='$articleTitle',"
. " article_styleElementText='$articleStyleElementText',"
. " article_body='$articleBody' WHERE article_id=$articleID";
```

5. Execute the SQL query, and cancel the redirect if there is an error.

```
DBQuery($query);
if (DBError() != ""){
    print DBError(); $redirect = false;
}
```

6. Disconnect from the database.

```
// Disconnect is required when updating
DBDisconnect(); }
```

7. If there have been no errors and the redirect flag is still set, redirect to *start.php*.

```
// redirect if no errors
if($redirect){ Header("Location: start.php"); } ?>
```

### Deleting an Article (delete.php and xt\_delete.php)

The file *delete.php* displays the chosen article information to allow the user to confirm that they wish to delete the specified article. Once confirmation is obtained, the relevant record in the database is deleted using the processing page, *xt\_delete.php*.

#### **delete.php**

1. Include *i\_database.php*.

```
<?
// include the database wrapper functions
require_once("./i_database.php");
```

2. Read the HTTP GET variable *article\_id* to find which article the user has requested to edit.

```
// get the article id
$articleID = $_HTTP_GET_VARS["article_id"];
```

3. Attempt to connect to the database; if there is an error, print it and cancel the script.

```
// Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
} else {
```

4. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the *articleID* variable is escaped for SQL.

```
// SQL query to get required document from the database
DBQuery("SELECT * FROM articles WHERE article_id = ". escape_string($articleID) . ";");
if (DBFetchRow()) {
?>
```

5. Create a standard HTML page, adding the heading "Delete a Document" and a message asking the user if they wish to delete the article.

```
<HTML>
<HEAD>
<TITLE>Delete Article ID <?=htmlspecialchars($articleID)?></TITLE>
```

```

</HEAD>
<BODY>
  <H1>Delete a Document</H1>
  <P>Are you sure you wish to delete this article?</P>

```

6. Create a table in the HTML page that displays the article id and title using the DBResult function.

```

<TABLE>
  <TR>
    <TD>Article ID:</TD>
    <TD><?=htmlspecialchars($articleID)?></TD>
  </TR>
  <TR>
    <TD>Title:</TD>
    <TD><?=htmlspecialchars(DBResult("article_title"))?></TD>
  </TR>
</TABLE>

```

7. Create a form in the page. Put the `xt_delete.php` file name in the action attribute of the FORM tag.

```

<FORM action="xt_delete.php" method="GET">

```

8. Add a hidden form object to the form to store the article id for processing.

```

<INPUT type="hidden" name="article_id" value="<?=htmlspecialchars($articleID)?>">

```

9. Add buttons to delete the article or cancel the deletion.

```

<P>
  <INPUT type="submit" value="Delete"> <INPUT type="button" value="Cancel"
    onclick="javascript:history.back();">
</P>

```

10. Close the form and document tags.

```

  </FORM>
</BODY>
</HTML>

```

11. If the article cannot be found, an error will need to be displayed.

```

<?
  } else {
    //There are no records matching this article_id
    //TODO: Handle and display meaningful error
  }
}

```

12. Disconnect from the database.

```

  DBDisconnect();
?>

```

### **xt\_delete.php**

1. Include `i_database.php` and initialize the redirect flag.

```

// include the database wrapper functions
require_once("../i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;

```

2. Attempt to delete the article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
if (!DBConnect()){
    print DBError();
    $redirect = false;
} else {
```

3. Attempt to delete the article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
DBQuery("DELETE FROM articles WHERE article_id = " . escape_string($articleID) . ";");

if (DBError() != "") {
    print DBError();
    $redirect = false;
}
```

4. Disconnect from the database.

```
// Disconnect is required when updating
DBDisconnect();
}
```

5. If there have been no errors and the redirect flag is still set, redirect to *start.php*.

```
// redirect if no errors
if($redirect){
    Header("Location: start.php");
}
?>
```

## Viewing an Article (view.php)

1. Include *i\_database.php*.

```
<?
// include the database wrapper functions
require_once("../i_database.php");
```

2. Read the HTTP GET variable *article\_id* to find which article the user has requested to edit.

```
// get the article id
$articleID = $_HTTP_GET_VARS["article_id"];
```

3. Attempt to connect to the database. If there is an error, print it and cancel the script.

```
// Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
} else {
```

4. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
//SQL query to get required document from the database
DBQuery("SELECT * FROM articles WHERE article_id = " . escape_string($articleID) . ";");
if (DBFetchRow()) {
?>
```



5. Use embedded PHP to both set the STYLE tag of the page to the article style information retrieved from the database and to display the article title and content.

```
<HTML>
  <HEAD>
    <TITLE><?=DBResult("article_title")?></TITLE>
    <STYLE> <?=DBResult("article_styleElementText")?> </STYLE>
  </HEAD>
  <BODY>
    <H1><?=DBResult("article_title")?></H1>
    <?=DBResult("article_body")?>
  </BODY>
</HTML>
```

6. If the article cannot be found, an error will need to be displayed.

```
<?
  } else {
    //There are no records matching this article_id
    //TODO: Handle and display meaningful error
  }
}
```

7. Disconnect from the database.

```
DBDisconnect();
?>
```

## Handling Image Uploads

To enable the image upload functionality, a new script must be created and then referenced in the configuration file.

To create and reference an upload script, refer to the [PHP HTTP File Upload Handler Script](#) section of the EditLive! SDK.

# MySQL in PHP Code

- [add.php](#)
- [delete.php](#)
- [edit.php](#)
- [i\\_database.php](#)
- [start.php](#)
- [view.php](#)
- [xt\\_add.php](#)
- [xt\\_delete.php](#)
- [xt\\_edit.php](#)

# add.php

```
<?
/*****
add.php -- create a new article

End users can enter content using a web form and EditLive!
The web form is submitted to the page xt_add.php

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

//load the XML file into the string "$xmlConfig"
// this helps to speed up the ELJ load time
$filename = "db_config.xml";
$fd = fopen($filename,"r");
$xmlConfig = fread ($fd, filesize($filename));
fclose ($fd);

//the initial content to load into ELJ
$content = "<p>This is the initial source</p>"

?>
<HTML>

<HEAD>
<meta HTTP-EQUIV="content-type" CONTENT="text/html; charset=UTF-8">
<TITLE>Create a New Article</TITLE>
<LINK rel="stylesheet" href="sample.css">

<?
// This script file initializes the EditLive! API so that the EditLive!
// properties and methods may be set to customise EditLive!.
?>
<script src="../../redistributables/editlivejava/editlivejava.js"></script>
</HEAD>
<BODY>

<H1>Create a New Article</H1>

<?
// This form contains EditLive!, a text area for the article title,
// a submit button and a cancel button.
?>
<FORM action="xt_add.php" method="POST" name="articleForm">

<?//Article title ?>
<P>Title: <INPUT type="text" name="article_title" size="50"></P>

<P>Body:<BR>

<script language="JavaScript">
var ELJApplet1_js;
ELJApplet1_js = new EditLiveJava("ELJApplet1", "700", "600");

ELJApplet1_js.setDownloadDirectory("../redistributables/editlivejava");
ELJApplet1_js.setConfigurationText("<?=rawurlencode($xmlConfig)?>");
ELJApplet1_js.setBody("<?=rawurlencode($content)?>");

ELJApplet1_js.setLocalDeployment(false);
ELJApplet1_js.setAutoSubmit(true);
ELJApplet1_js.setDebugLevel("info");
```

```
ELJApplet1_js.setShowSystemRequirementsError(true);
ELJApplet1_js.show();
</script>
</P>

<P><INPUT type="submit" value="Save" name="Add Article">
<INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>

</FORM>

</BODY>
</HTML>
```

# delete.php

```
<?
/*****
delete.php -- confirm the user wants to delete an article

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

// include the database wrapper functions
require_once("./i_database.php");

// get the article id
$articleID = $_HTTP_GET_VARS["article_id"];

// Attempt to establish a connection with the database
if (DBConnect() == FALSE) {
    print DBError();
} else {
    //SQL query to get required document from the database
    DBQuery("SELECT * FROM articles WHERE article_id = " . escape_string($articleID) . ";" );
    if (DBFetchRow()) {

//Display the article information in a form asking for delete confirmation
?>

<HTML>
<HEAD>
<TITLE>Delete Article ID <?=htmlspecialchars($articleID)?></TITLE>
<LINK rel="stylesheet" href="sample.css">

</HEAD>
<BODY>

<H1>Delete a Document</H1>

<P>Are you sure you wish to delete this article?</P>

<TABLE>
<TR>
<TD>Article ID:</TD>
<TD><?=htmlspecialchars($articleID)?></TD>
</TR>
<TR>
<TD>Title:</TD>
<TD><?=htmlspecialchars(DBResult("article_title"))?></TD>
</TR>
</TABLE>

<FORM action="xt_delete.php" method="GET">
<INPUT type="hidden" name="article_id" value="<?=htmlspecialchars($articleID)?>">
<P><INPUT type="submit" value="Delete"> <INPUT type="button" value="Cancel" onclick="javascript:history.back();"
></P>
</FORM>
</BODY>
</HTML>
<?
    } else {
        //There are no records matching this article_id
        //TODO: Handle and display meaningful error
    }
}
```

```
DBDisconnect();  
?>
```

# edit.php

```
<?
/*****
edit.php -- update an existing article

End users can enter content using a web form and EditLive!
The web form is submitted to the page xt_edit.php

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

//load the XML file into the string "$xmlConfig"
// this helps to speed up the ELJ load time
$filename = "db_config.xml";
$fd = fopen($filename,"r");
$xmlConfig = fread ($fd, filesize($filename));
fclose ($fd);

// include the database wrapper functions
require_once("./i_database.php");

// get the article id
$articleID = $HTTP_GET_VARS["article_id"];

// Attempt to establish a connection with the database
if (DBConnect() == FALSE) {
    print DBError();
} else {
    ?>

<HTML>
<HEAD>
<meta HTTP-EQUIV="content-type" CONTENT="text/html; charset=UTF-8">
<TITLE>Editing Article ID <?=$articleID?></TITLE>
<LINK rel="stylesheet" href="sample.css">

</HEAD>
<BODY>

<?
    //SQL query to get required document from the database
    DBQuery("SELECT * FROM articles WHERE article_id = " . escape_string($articleID) . " ");
    if (DBFetchRow()) {
// This form contains EditLive!, a text area for the article title,
// a submit button and a cancel button.
?>

<H1>Edit Document</H1>

<FORM name="form1" method="post" action="xt_edit.php">

<?// Hidden field for identifying the article?>
<INPUT type="hidden" name="article_id" value="<?=DBResult("article_id")?>">

<?// Article title?>
<P>Title: <INPUT type="text" name="article_title" value="<?=htmlspecialchars(DBResult("article_title"))?>"
size="40"></P>

<P>Body:
<script src="../../redistributables/editlivejava/editlivejava.js"></script>
```

```

<script language="JavaScript">
var ELJApplet1_js;
ELJApplet1_js = new EditLiveJava("ELJApplet1", "700", "600");

ELJApplet1_js.setDownloadDirectory(".././redistributables/editlivejava");
ELJApplet1_js.setConfigurationText("<?rawurlencode($xmlConfig)?>");
ELJApplet1_js.setBody("<?rawurlencode(DBResult("article_body"))?>");
ELJApplet1_js.setStyles("<?rawurlencode(DBResult("article_styleElementText"))?>");

ELJApplet1_js.setLocalDeployment(false);
ELJApplet1_js.setAutoSubmit(true);
ELJApplet1_js.setDebugLevel("info");
ELJApplet1_js.setShowSystemRequirementsError(true);
ELJApplet1_js.show();
</script>

<P><INPUT type="submit" value="Save"> <INPUT type="button" value="Cancel" onclick="javascript:history.back();"><
/P>

</FORM>
<?
    } else {
        echo "No record found";
        //There are no records matching this article_id
        //TODO: Handle and display meaningful error
    }
}
DBDisconnect();
?>
</BODY>
</HTML>

```



# i\_database.php

```
<?php
/*****
 *
 * i_database.php
 *
 * This file contains functions to connect to and retrieve content from
 * the sample database.
 *
 *****/
$DBLink = null;
$DBResult = null;
$i = 0;

// Connect to the defined database, see the documentation for how to set "ELContent" up
function DBConnect() {
    global $DBLink;

    $DBName = "ELContent";
    $hostname = "localhost";
    /*****
     * Change these lines to set your database username and password *
     *****/
    $username = "";
    $password = "";

    // Use persistent connect as this reduces latency and increases overall efficiency
    $DBLink = mysql_pconnect($hostname,$username,$password);
    if ($DBLink) {
        return(mysql_select_db($DBName));
    }
    return false;
}

// Disconnect from the database, only used when updating as we are connecting with pconnect
// which leaves the connection open even after this is called
function DBDisconnect() {
    global $DBLink;

    mysql_close($DBLink);
}

// Execute an SQL query on the database
// it is assumed this string is safe, using the escape_string before passing here
function DBQuery($SQLQuery) {
    global $DBLink;
    global $DBResult;
    global $i;

    $DBResult = mysql_query($SQLQuery);
    $i = 0;
    return($DBResult);
}

// Fetch the next row, returning true if it exists otherwise false
function DBFetchRow() {
    global $DBResult;
    global $i;

    if ($i < mysql_num_rows($DBResult)) {
        $success = mysql_data_seek($DBResult,$i);
        $i++;
    } else {
        $success = false;
    }
}
```

```

        return $success;
    }

    // Retrieve a column (specified by number or name) from the current row
    function DBResult($column) {
        global $DBResult;
        global $i;

        return(mysql_result($DBResult, $i - 1, $column));
    }

    // Retrieve the last Database Error, returns an empty string otherwise
    function DBError() {
        return(mysql_error());
    }

    // escape the string for MySQL usage
    function escape_string($string) {
        global $DBLink;

        // make sure it is not escaped already
        $string = stripslashes($string);

        if(version_compare(PHP_VERSION, "4.3.0") <= "-1") {
            return(mysql_escape_string($string));
        } elseif($DBLink) {
            return(mysql_real_escape_string($string, $DBLink));
        }
    }
}
?>

```

# start.php

```
<?php
/*****

start.php -- the home page for the application

Lists the articles stored in the database and provides
links for creating a new article, editing existing articles
and deleting existing articles

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

// include the database wrapper functions
require_once("./i_database.php");
?>

<HTML>

<HEAD>
<TITLE>Sample Database Application for PHP</TITLE>
<LINK rel="stylesheet" href="sample.css">
</HEAD>

<BODY>

<H1>Sample Database Application for PHP</H1>

<P><A href="add.php">Create a new article</A></P>

<?
//Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
} else {
    DBQuery("select * from articles");
    if (DBFetchRow()) {
?>

<TABLE cellpadding=3 cellspacing=0 border=0>
  <TR>
    <TH>ID</TH>
    <TH width="250">Title</TH>
    <TH>Available Actions</TH>
  </TR>
  <?
      // There are records. Loop through all the records in the
      // recordset and write out a table row for each record
      do {
?>
    <TR>
      <TD><?=DBResult("article_id")?></TD>
      <TD><?=DBResult("article_title")?></TD>
      <TD><A href="view.php?article_id=<?=DBResult("article_id")?>">View</A> |
        <A href="edit.php?article_id=<?=DBResult("article_id")?>">Edit</A> |
        <A href="delete.php?article_id=<?=DBResult("article_id")?>">Delete</A>
      </TD>
    </TR>
  <?
      } while(DBFetchRow())
?>
</TABLE>

<?
} else {
?><P>There are no records in the database. Click <STRONG>Create a new article</STRONG>
```

```
to add a record to the database.</P>
<?
  }
}
DBDisconnect();
?>

</BODY>

</HTML>
```

# view.php

```
<?
/*****
view.php -- template for displaying an article

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

// include the database wrapper functions
require_once("./i_database.php");

// get the article id
$articleID = $HTTP_GET_VARS["article_id"];

// Attempt to establish a connection with the database
if (DBConnect() == FALSE) {
    print DBError();
} else {
    //SQL query to get required document from the database
    DBQuery("SELECT * FROM articles WHERE article_id = " . escape_string($articleID) . ";");
    if (DBFetchRow()) {
?>
<HTML>
<HEAD>
<meta HTTP-EQUIV="content-type" CONTENT="text/html; charset=UTF-8">
<TITLE><?=DBResult("article_title")?></TITLE>
<STYLE>
<?=DBResult("article_styleElementText")?>
</STYLE>
</HEAD>
<BODY>

<H1><?=DBResult("article_title")?></H1>

<?=DBResult("article_body")?>

</BODY>
</HTML>
<?
    } else {
        //There are no records matching this article_id
        //TODO: Handle and display meaningful error
    }
}
DBDisconnect();
?>
```

# xt\_add.php

```
<?
/*****

xt_add.php -- add a record to the database

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

// include the database wrapper functions
require_once("./i_database.php");

// redirect variable, we don't want to redirect if we have errors
$redirect = TRUE;

//Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
    $redirect = FALSE;
} else {
    // Get the POSTed data from the form and escape it for SQL using the function in i_database
    $articleTitle = escape_string($_HTTP_POST_VARS["article_title"]);
    $articleStyleElementText = escape_string($_HTTP_POST_VARS["ELJApplet1_styles"]);
    $articleBody = escape_string($_HTTP_POST_VARS["ELJApplet1"]);

    // Insert the POSTed data into the database
    $query = "INSERT INTO articles ( article_title, article_styleElementText, article_body ) "
        . "VALUES ( '$articleTitle', '$articleStyleElementText', '$articleBody' )";
    DBQuery($query);
    if (DBError() != "") {
        print DBError();
        $redirect = FALSE;
    }

    // Disconnect is required when updating
    DBDisconnect();
}

// redirect if no errors
if($redirect){
    Header("Location: start.php");
}
?>
```

## xt\_delete.php

```
<?
/*****

xt_delete.php -- delete the specified article from the database

article_id passed in through QueryString

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement
*****/

// include the database wrapper functions
require_once("./i_database.php");

// get the article id
$articleID = $HTTP_GET_VARS["article_id"];

// redirect variable, we don't want to redirect if we have errors
$redirect = TRUE;

//Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
    $redirect = FALSE;
} else {
    DBQuery("DELETE FROM articles WHERE article_id = " . escape_string($articleID) . ";");
    if (DBError() != "") {
        print DBError();
        $redirect = FALSE;
    }

    // Disconnect is required when updating
    DBDisconnect();
}

// redirect if no errors
if($redirect){
    Header("Location: start.php");
}
?>
```

# xt\_edit.php

```
<?
/*****

xt_edit.php -- add the article to the database

Receives content from edit.asp

Copyright (c) 2005 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/
// include the database wrapper functions
require_once("./i_database.php");

// redirect variable, we don't want to redirect if we have errors
$redirect = TRUE;

//Attempt to establish a connection with the database
if (!DBConnect()) {
    print DBError();
    $redirect = FALSE;
} else {
    // Get the POSTed data from the form and escape it for SQL using the function in i_database
    $articleID = escape_string($_HTTP_POST_VARS["article_id"]);
    $articleTitle = escape_string($_HTTP_POST_VARS["article_title"]);
    $articleStyleElementText = escape_string($_HTTP_POST_VARS["ELJApplet1_styles"]);
    $articleBody = escape_string($_HTTP_POST_VARS["ELJApplet1"]);

    // Insert the POSTed data into the database
    $query = "UPDATE articles SET article_title='$articleTitle',
article_styleElementText='$articleStyleElementText',
    . " article_body='$articleBody' WHERE article_id=$articleID";
    DBQuery($query);
    if (DBError() != "") {
        $redirect = FALSE;
    }

    // Disconnect is required when updating
    DBDisconnect();
}

// redirect if no errors
if($redirect){
    Header("Location: start.php");
}
?>
```



# Automated Plugin Loading PHP Example

## Documentation

The [Automated Plugin Loading PHP Documentation](#) provides a walk-through on how the example is created.

## Code

The complete code views for all the associated files in the Automated Plugin Loading PHP Example are available [here](#).

# Automated Plugin Loading PHP Documentation

This page provides information on how a JSP script can be used to automatically load plugins into EditLive!. For more information on plugins, see the [Creating and Using Plugins in the Applet](#) article in the [Developer Guide](#) for this SDK.

## Getting Started

### Required License

EditLive!'s [Advanced API](#) and [Plugin](#) functionality is only supported if an EditLive! [Enterprise Edition](#) license has been installed for the editor or if the user is still within their 30 day trial period.

### Required Skills

The following skills are required prior to working with this sample:

- Creating reusable functions in Java Server Pages
- Basic client-side JavaScript

### Set Up Your Server

Ensure you have set up your Web server for Java server-side processing, as described in the EditLive! [Install Guide](#).

## Overview

In this example, EditLive! is created using the Javascript [Load Time Methods](#). A function is called in a JSP script which will write out additional load-time properties to instantiate each plugin located in a specific directory.

A database is not required for this example. You can not perform any saving of document content in this example due to the lack of server-side processing in the example code and the absence of a database.

This sample demonstrates how to perform the following with EditLive! and JSP:

- Iterate through all plugins located in a specific directory and dynamically call to the EditLive! load-time properties to load each plugin.

## JSP Script for Automatically Adding Plugins

### pluginLoader.jsp

The pluginLoader.jsp script contains a Java function called loadPlugins. loadPlugins contains a routine for searching through a specified directory, locating any instances of .xml files. If an .xml file is found, a call to the [addPluginAsText property](#) for EditLive! is written to the page.

loadPlugins requires the following parameters:

- **out** - The JspWriter instance used by the ServletResponse to write to the webpage.
- **elName** - The name of the javascript variable used to specify load-time and run-time properties of EditLive!
- **pluginsDirectory** - The path to the directory on the server containing all of the desired EditLive! plugins.
- **pluginsURL** - The URL for the directory on the server containing all of the desired plugins.

## Initializing EditLive! for Java and Automatically Loading the Plugins

### example.jsp

To embed EditLive! within a Web page and automatically load all plugins found in a specified directory, several steps are required. Each of these steps is explained here with code samples provided.

1. Create an instance of EditLive! using the javascript [Load Time Methods](#).

```
<html>
  <head>
    <title>Automated Plugin Loading Example - JSP</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Automated Plugin Loading Example</h1>
```

```

<p>This example depicts how a JSP script can be used to add all of the plugins located
in a specific to an EditLive! instance.</p>

<!--
The instance of EditLive!
-->
<script language="JavaScript">
    // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
    var editlive = new EditLiveJava("ELApplet", 700, 400);

    // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../..//redistributables/editlivejava/sample_eljconfig.
xml");

    // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
    // at the this location.
    editlive.show();
</script>
</body>
</html>

```

2. Define the Content type for the page and created a reference to the pluginLoader.jsp script.

```

<%@page contentType="text/html"%>

<%@ include file="pluginLoader.jsp" %>

<html>
    <head>
        <title>Automated Plugin Loading Example - JSP</title>
        <link rel="stylesheet" href="stylesheet.css">
        <!--
        Include the EditLive! JavaScript Library
        -->
        <script src="../..//redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
    </head>
    <body>

        <h1>Automated Plugin Loading Example</h1>

        <p>This example depicts how a JSP script can be used to add all of the plugins located
in a specific to an EditLive! instance.</p>

        <!--
        The instance of EditLive!
        -->
        <script language="JavaScript">
            // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
            var editlive = new EditLiveJava("ELApplet", 700, 400);

            // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../..//redistributables/editlivejava/sample_eljconfig.
xml");

            // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
            // at the this location.
            editlive.show();
        </script>
    </body>
</html>

```

3. Create the Strings representing the path and the URL to the plugins directory on the server.

```

<%@page contentType="text/html"%>

<%@ include file="pluginLoader.jsp" %>

<html>
  <head>
    <title>Automated Plugin Loading Example - JSP</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Automated Plugin Loading Example</h1>

    <p>This example depicts how a JSP script can be used to add all of the plugins located
    in a specific to an EditLive! instance.</p>

    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
      a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative or absolute path to the XML configuration file to use
      editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      <%
        String pluginsDir = getServletContext().getRealPath("/examplePlugins/");
        String pluginsURL = request.getScheme() + "://" + request.getServerName() + ":"
+ request.getServerPort() + request.getContextPath() + "/examplePlugins/";
      %>

      // .show is the final call and instructs the JavaScript library (editlivejava.js) to
      insert a new EditLive! instance
      // at the this location.
      editlive.show();
    </script>
  </body>
</html>

```

4. Call to the loadPlugins function (located in the pluginLoader.jsp script), passing the required parameters.

```

<%@page contentType="text/html"%>

<%@ include file="pluginLoader.jsp" %>

<html>
  <head>
    <title>Automated Plugin Loading Example - JSP</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Automated Plugin Loading Example</h1>

    <p>This example depicts how a JSP script can be used to add all of the plugins located
    in a specific to an EditLive! instance.</p>

    <!--
    The instance of EditLive!

```

```

-->
<script language="JavaScript">
// Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
var editlive = new EditLiveJava("ELApplet", 700, 400);

// This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../..redistributables/editlivejava/sample_eljconfig.
xml");

    <%
        String pluginsDir = getServletContext().getRealPath("/examplePlugins/");
        String pluginsURL = request.getScheme() + "://" + request.getServerName() + ":"
+ request.getServerPort() + request.getContextPath() + "/examplePlugins/";

        loadPlugins(out, "editlive", pluginsDir, pluginsURL);
    %>

// .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
// at the this location.
editlive.show();
</script>
</body>
</html>

```

## Summary

Using a server-side script, developers can store all desired plugins in a single directory and ensure each plugin is loaded with EditLive!. This solution can cut down on maintenance times required for continually hand-coding load-time properties for each new plugin desired to function with EditLive!.

# Automated Plugin Loading PHP Code

- [example.php](#)
- [pluginLoader.php](#)

# example.php

```
<?php
/*****

example.php -- A standard javascript implementation of EditLive!
                A call is made to the addPlugins PHP method to
                add plugins to the EditLive! instance.

This is an example of how a PHP script can be used to easily
add plugins from a specified directory to EditLive!.

Copyright (c) 2007 Ephox Corporation. All rights reserved.
See license.txt for license agreement

*****/

require_once("./pluginLoader.php");
?>
<html>
  <head>
    <title>Automated Plugin Loading Example - PHP</title>
    <link rel="stylesheet" href="stylesheet.css">
    <!--
    Include the EditLive! JavaScript Library
    -->
    <script src="../../redistributables/editlivejava/editlivejava.js" language="JavaScript"></script>
  </head>
  <body>

    <h1>Automated Plugin Loading Example</h1>

    <p>This example depicts how a PHP script can be used to add all of the plugins located
in a specific to an EditLive! instance.</p>
    <!--
    The instance of EditLive!
    -->
    <script language="JavaScript">
      // Create a new EditLive! instance with the name "ELApplet", a height of 400 pixels and
a width of 700 pixels.
      var editlive = new EditLiveJava("ELApplet", 700, 400);

      // This sets a relative path to the directory where the EditLive! redistributables can
be found e.g. editlivejava.jar
      editlive.setDownloadDirectory("../../redistributables/editlivejava");

      // This sets a relative or absolute path to the XML configuration file to use
editlive.setConfigurationFile("../../redistributables/editlivejava/sample_eljconfig.
xml");

      // this load-time property is only required to enable Equation Editing
      editlive.setUseMathML(true);

    </script>
  </body>
</html>
?>
$protocol = 'http';
if (isset($_SERVER['SERVER_PORT']) && $_SERVER['SERVER_PORT'] == '443') {
    $protocol = 'https';
}

$hostname = $protocol.'://'.$_SERVER['HTTP_HOST'];
$dirname = $_SERVER['PHP_SELF'];
$dirname = substr($dirname, 0, strlen($dirname)-24)."examplePlugins/";
$hostname = $hostname.$dirname;

$pluginPath = substr(__FILE__, 0, strlen(__FILE__)-24)."examplePlugins";

loadPlugins("editlive", $pluginPath, $hostname);
?>
```

```
        // .show is the final call and instructs the JavaScript library (editlivejava.js) to
insert a new EditLive! instance
        // at the this location.
        editlive.show();
    </script>
</body>
</html>
```



# pluginLoader.php

```
<?php
function loadPlugins($jsVariableName, $pluginDir, $pluginUrl) {
    if (is_dir($pluginDir) && $dh = opendir($pluginDir)) {
        while (($file = readdir($dh)) != false) {
            $extension = substr($file, strlen($file) - 4, strlen($file));

            if(strcmp(".xml", $extension) == 0) {;
                $filename = $pluginDir . '/' . $file;
                $content = file_get_contents($filename);
                if ($content) {
                    $content = rawurlencode($content);
                    echo "$jsVariableName.addPluginAsText('$content', '$pluginUrl');\n";
                }
            }
        }
        closedir($dh);
    }
}
?>
```

# IBM Web Content Management Integration

IBM OEM Support



**Please note:** If you have obtained EditLive for IBM WCM directly from IBM you will need to lodge support requests exclusively with IBM and not Ephox. This will ensure your issues are addressed in the most expedient fashion and that you only need to work with one vendor.

- [IBM Web Content Management 7+](#)
- [IBM Web Content Management 6.1+](#)
- [IBM Web Content Management 6.1](#)
- [Specifying Configurations in IBM WCM](#)
- [Licensing EditLive IBM WCM](#)
- [Troubleshooting in IBM WCM](#)
- [Uninstalling the IBM WCM Integration](#)

This integration is a coupling of Ephox client-side content authoring software with the IBM Web Content Management (WCM) architecture. This integration allows current designers and users of the WCM environment to enjoy an intuitive and comprehensive interface for entering rich text into their content.

This integration is compatible with IBM Websphere 6.1 and above.

This document serves as a comprehensive knowledge base for all information related to the integration between Ephox EditLive! and the IBM WCM architecture. There is detailed information throughout this document on the purpose and basic structure of this integration and step-by-step instructions to install this integration. For more information on EditLive!, see the [EditLive! Developers Guide](#) on the [Ephox website](#).

## Integration Purpose

The goal of this integration is to provide greater content creation and editing abilities to users and designers of the WCM environment. On completion of this integration, WCM users will be able to access the content of a Rich Text field through a single interface that provides support for the following rich text elements:

- Tables - specify rows, columns, horizontal and vertical alignment, cell spacing, and padding.
- Spell Checking - EditLive! comes packaged with 13 different language dictionaries.
- Specified CSS Rendering - Developers can specify the CSS to load into EditLive!, to ensure all content is created in a true wysiwyg environment.
- W3C and Section 508 Accessibility Checking - Users can check their content with the EditLive! accessibility tool to ensure compliance with a variety of accessibility standards.

When integrated into the WCM architecture, EditLive! enables different configurations of the rich text editor to be used depending on the current user's access permissions. This allows each user involved in a document's work flow to only use the functionality permitted in EditLive!.

Each time EditLive! is loaded, it requires a reference to an EditLive! configuration file. This file is responsible for specifying a variety of visual and functional components of the EditLive! editor. For more information on creating a configuration file for EditLive!, see the [EditLive! Developer Guide](#) packaged with this integration.

An example would be an WCM instance used by a newspaper publishing organization. Users belonging to the Journalist user group may only be able to use rich text styles and hyperlinks. Editor users won't need to create hyperlinks, but would need rich text displays, such as color, to create changes and comments that stand out to Journalists. Designer users would require the image insertion and tables functionalities, but won't need dictionaries or hyperlinks enabled.

Information on how to set up role-based configuration of EditLive! can be found in the [Specifying EditLive! Configurations](#) section of this documentation. An example on how to customize EditLive! based on user and/or user group is also available in the `ephox.config.properties.sample` file (located in `res/editlivejava`).

# IBM Web Content Management 7+

- [Installing EditLive IBM WCM 7+](#)
- [Upgrading the Integration IBM WCM 7+](#)
- [Additional Configuration Items](#)

# Installing EditLive IBM WCM 7+

The EditLive! WCM integration is packaged as a standard JEE EAR file. To install the EditLive! integration you can use either the IBM Integrated Solutions Console or your own installation process. The documentation below outlines how to perform an installation using the WebSphere Application Server.

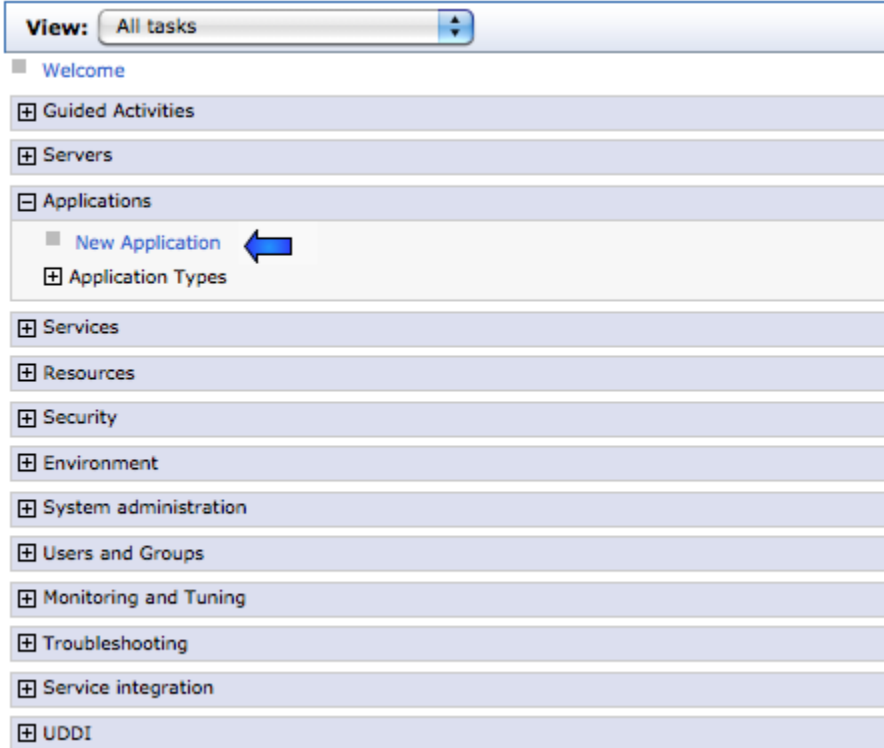
## Step 1 - Install the EAR file

The server running WebSphere\_Portal needs to be started before you begin the install process.

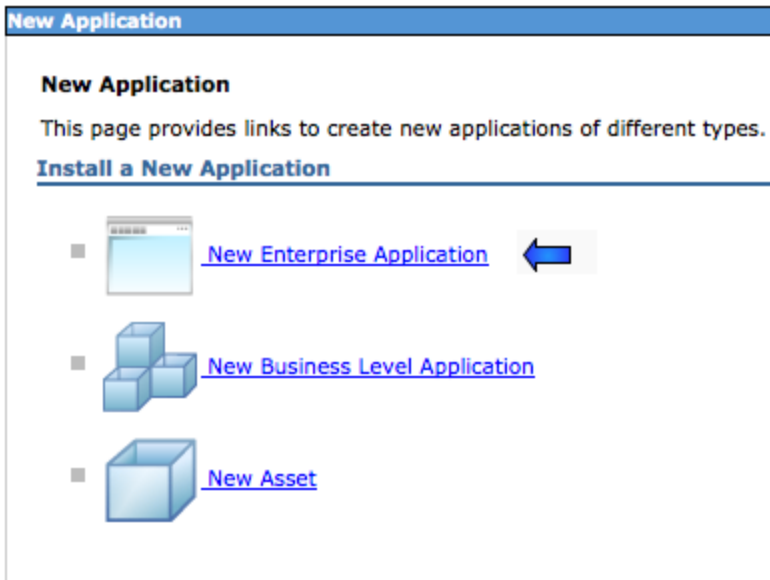
Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation, the url for the console will be: WebSphere Application Server 7.0 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to the New Application link in the console. This will involve clicking on the "Install New Application" link.



2. Select the New Enterprise Application link.



3. From the "Choose Application" form, choose the new version of the *EphoxEditLiveJEE.ear* file and click on "Next".

Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

Local file system

Full path  
x\_LWCM\_6.1/EphoxEditLiveJEE.ear Browse...

Remote file system

Full path  
Browse...

Next Cancel

4. Select the Fast Path install option and click the Next button.

Preparing for the application installation

How do you want to install the application?

Fast Path - Prompt only when additional information is required.

Detailed - Show all installation options and parameters.

Choose to generate default bindings and mappings

Previous Next Cancel

5. You now will begin the wizard for installing a new application. The EAR file has had been packaged to run well with as many default options as possible.

- In the first step the defaults are all suitable. Simply click the "Next" button.

Enterprise Applications

**Install New Application**

Specify options for installing enterprise applications and modules.

**Step 1: Select installation options**

Step 2 Map modules to servers

Step 3 Map virtual hosts for Web modules

Step 4 Summary

**Select installation options**

Specify the various options that are available to prepare and install your application.

Precompile JavaServer Pages files

Directory to install application

Distribute application

Use Binary Configuration

Deploy enterprise beans

Application name

Create MBeans for resources

Enable class reloading

Reload interval in seconds

Deploy Web services

Validate Input off/warn/fail

Process embedded configuration

**File Permission**

Allow all files to be read but not written to  
 Allow executables to execute  
 Allow HTML and image files to be read by everyone

Application Build ID

Allow dispatching includes to remote resources

Allow servicing includes from remote resources

- In step 2, you will need to ensure that the modules are deployed to the server running WebSphere Portal. Typically this will be the server called "WebSphere\_Portal". If this server does not appear in the list, it may not have been started yet. To ensure that the ear is deployed to the right location, you will need to select the server, select the module, and then click the "Apply" button.

Enterprise Applications

**Install New Application**

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

**Step 2: Map modules to servers**

Step 3 Map virtual hosts for Web modules

Step 4 Summary

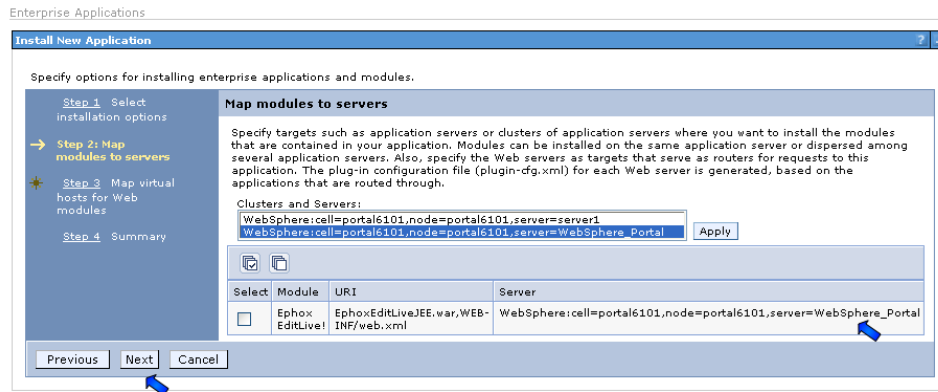
**Map modules to servers**

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

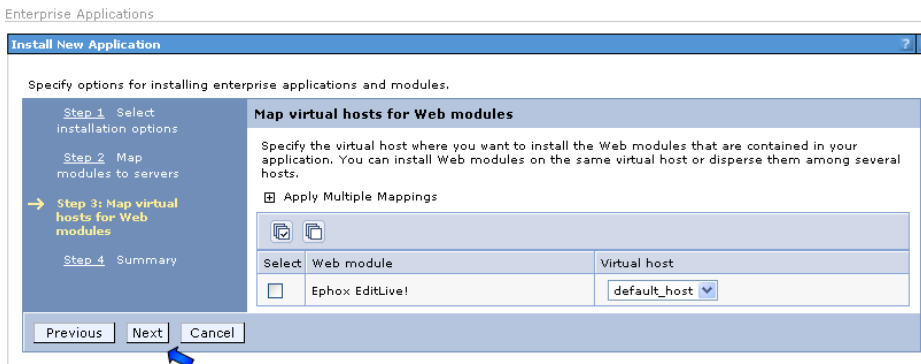
Clusters and Servers:

Select	Module	URI	Server
<input checked="" type="checkbox"/>	EphoxEditLive!	EphoxEditLiveJEE.war,WEB-INF/web.xml	WebSphere:cell=portal6101,node=portal6101,server=server1

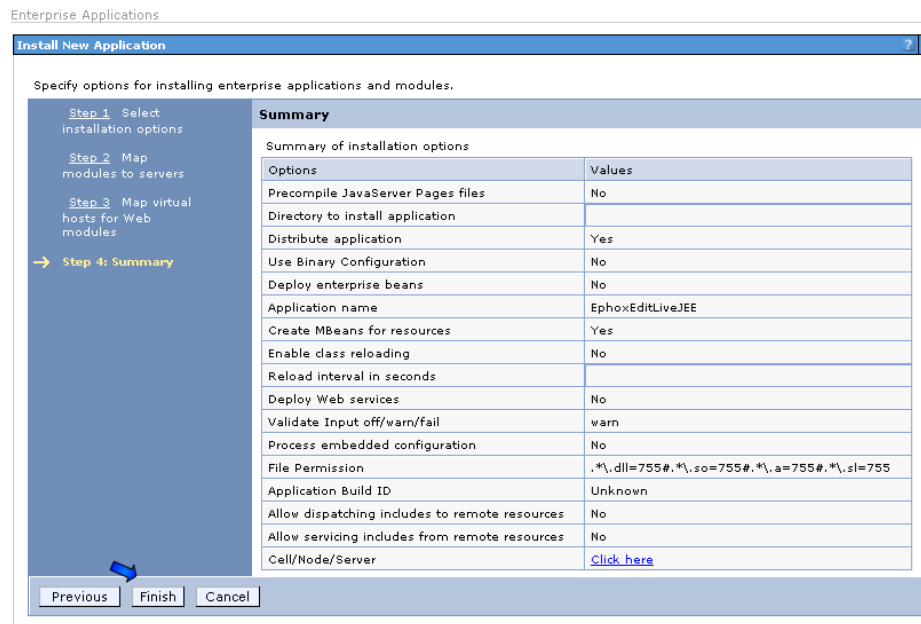
After doing this, you will expect the server section to look as indicated in the screenshot below. You can then press the "Next" button to go to step 3.



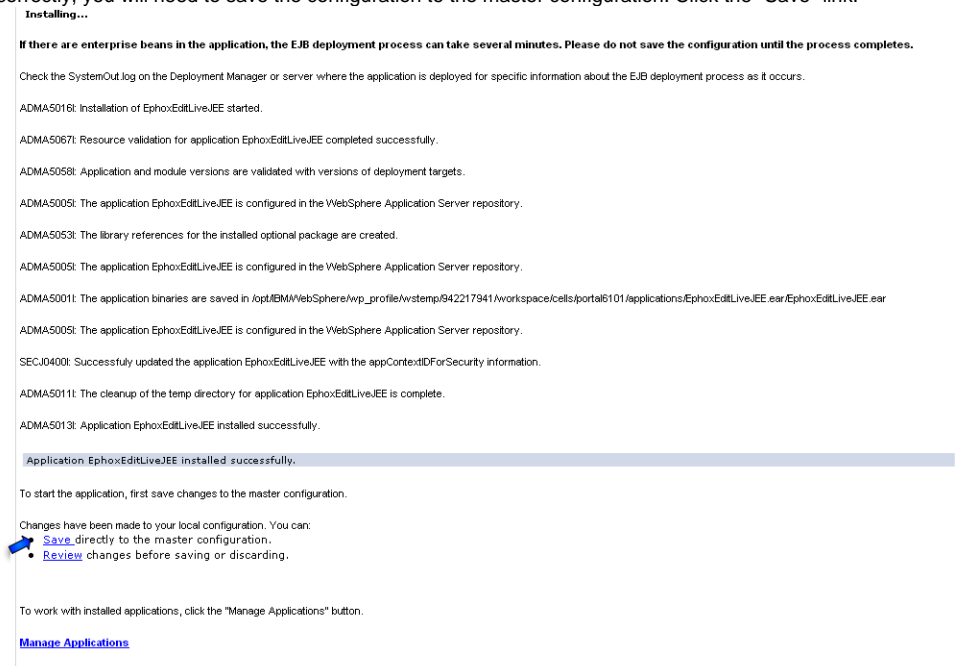
- The default options in Step 3 should be suitable, so you will typically be able to click the "Next" button and continue the wizard.



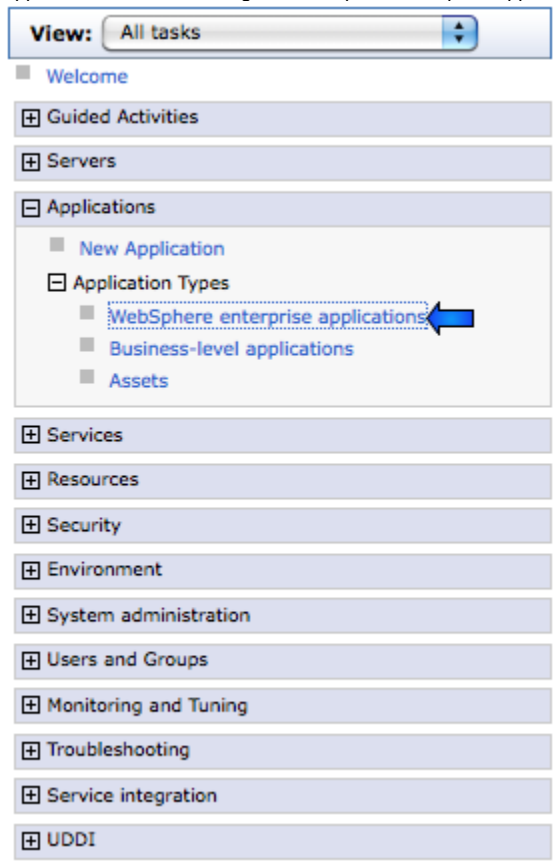
- In step 4 you are presented with a summary. After confirming the details, click "Finish" to complete the installation.



6. After finishing the wizard, the installation will take place, presenting the following in your browser. To ensure that the changes are applied correctly, you will need to save the configuration to the master configuration. Click the "Save" link.

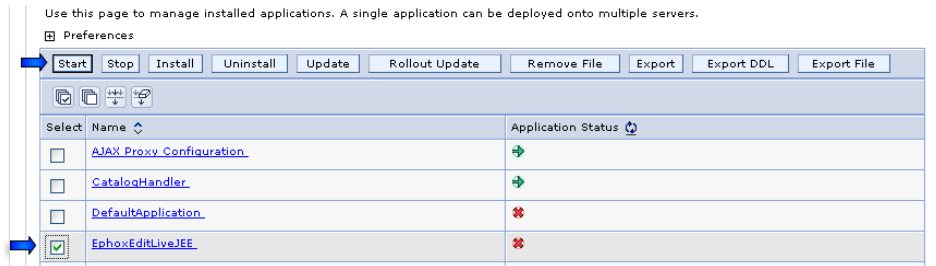


7. After installing the application, you will need to start it up. This can be done via the manage applications screen. First, navigate to the manage applications screen, clicking the WebSphere enterprise applications link.

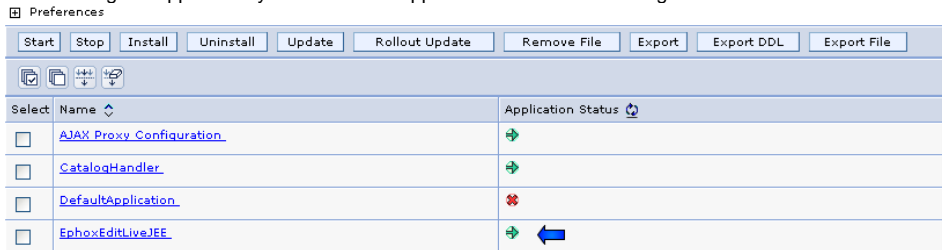


Then select the EphoxEditLiveJEE application and click the "Start" link.



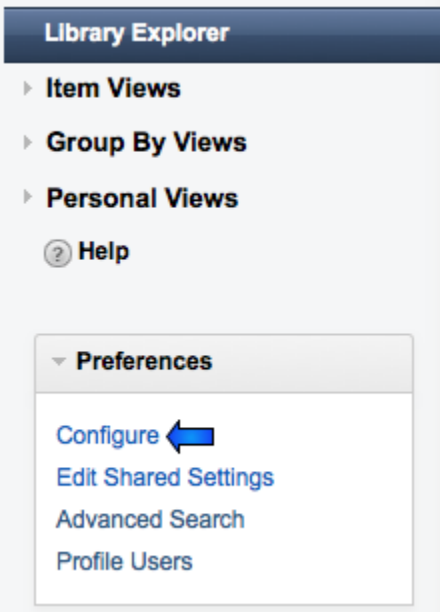


After starting the application you will see the application status now has a green arrow.

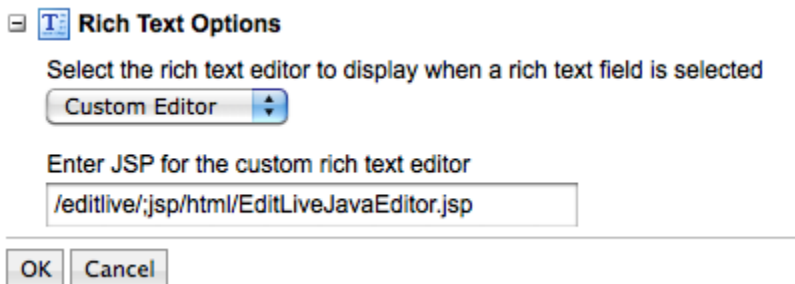


## Step 2 - Specify the EditLiveJavaEditor.jsp File in WCM

1. Log into Websphere Portal as a user with administrative WCM permissions.
2. Navigate to the Web Content Management Authoring Portlet.
3. Expand the Preferences link and click the Configure link.



4. Expand the Rich Text Options section.
5. Select Custom Editor from the drop-down.
6. In the text field displayed, enter `/editlive/;jsp/html/EditLiveJavaEditor.jsp` (The semi-colon is not a typo).



7. Click OK.

If you haven't clicked the Configure link for WCM before, this may cause the current default Library associated with the WCM portlet to be removed. To reassign the default Library assign to the WCM portlet, once inside the configure page expand the Library tab, select Web Content, and click the Add button.

## Client-Side Requirements

The EditLive! [Client-Side Requirements](#) can be found in the [Install Guide](#).

Although EditLive! itself supports a variety of JRE versions, WCM itself only supports a subset of the available JRE versions.

Visit the [WCM 7.0 Documentation](#) for an up to date list of the JRE versions supported by WCM.

# Upgrading the Integration IBM WCM 7+

## General Upgrade Procedure

To upgrade to a new release of the Tiny integration into WCM, perform the following steps:

- If you have made changes to *sample\_eljconfig.xml* or *customFunctionality.jsp* in order to use role based configurations, you'll need to back up these files by copying them to a location outside of your EditLive! installation.
- Follow the first step of the [Installation Instructions](#) (extract the zip file).
- Merge any changes you made from *sample\_eljconfig.xml* or *customFunctionality.jsp* into the copy of those files in the newly extracted folder.

## Step 1 - Update the EAR file

The server running WebSphere\_Portal needs to be started before you begin the update process.

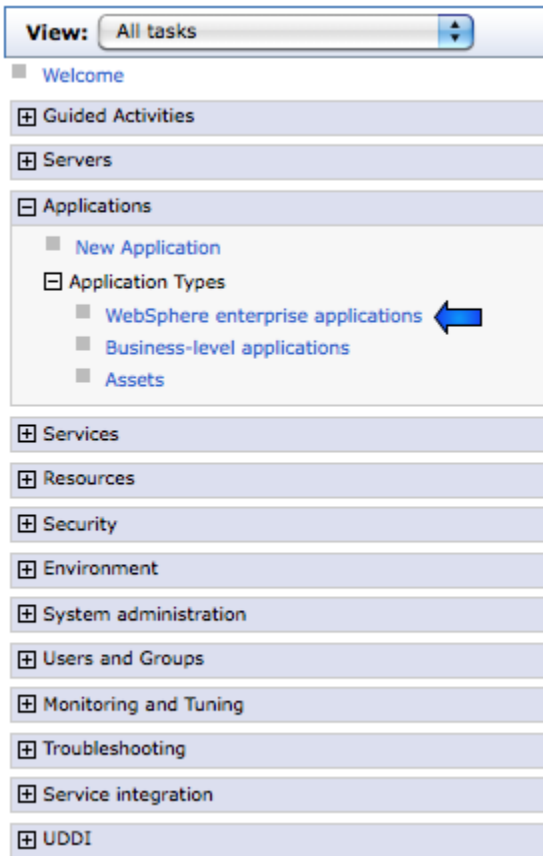
Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation the urls for the console will be:

WebSphere Application Server 6.1 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

WebSphere Application Server 7.0 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to Enterprise Applications in the console. To do this, click on WebSphere enterprise applications link.



2. A list of installed applications will be displayed. Select the "EphoxEditLiveJEE" application from the list and press the "Stop" button.

**Enterprise Applications**

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Preferences

Start Stop **Install** Uninstall Update Rollout Update Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➔
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➔
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dojo_Resources</a>	➔
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➔
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	➔
<input type="checkbox"/>	<a href="#">FeedService</a>	➔
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➔
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➔
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➔
<input type="checkbox"/>	<a href="#">LWP_People</a>	➔
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➔
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➔
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➔
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➔
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➔
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➔
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➔
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➔
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➔

Page: 1 of 6 Total 114

- The page will reload and EphoxEditLiveJEE will now display an X in its "Application Status." Select this application again and press the "Update" button.

**Enterprise Applications**

Messages  
Application EphoxEditLiveJEE on server WebSphere\_Portal and node ephox stopped successfully.

**Enterprise Applications**  
Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Preferences

Start Stop Install Uninstall Update **Rollout Update** Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➔
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➔
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dolo_Resources</a>	➔
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➔
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	✖
<input type="checkbox"/>	<a href="#">FeedService</a>	➔
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➔
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➔
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➔
<input type="checkbox"/>	<a href="#">LWP_People</a>	➔
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➔
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➔
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➔
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➔
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➔
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➔
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➔
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➔
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➔

Page: 1 of 6 Total 114

- The next page will provide a list of application update options. The first of these is "Replace the entire application." Ensure that this option is selected and then specify the path to the latest version of the EditLive! ear file, which can either be on a local or remote file system.

Application to be updated:

EphoxEditLiveJEE

**Application update options**

Replace the entire application  
Upload an enterprise archive (\*.ear) to replace the entire installed application.

**Specify the path to the replacement ear file.**

Local file system  
Full path  
/Users/dev/Desktop/Ephox\_LWCM

Remote file system  
Full path

Context root  
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

**How do you want to install the application?**

Prompt me only when additional information is required.  
 Show me all installation options and parameters.

5. After you have finished the "Replace the entire application" section, navigate to the bottom of the page and choose "Next".

6. From this point on, installation can be completed by following Step 1 of the [Installation Instructions](#), starting from point #4. Please note that you will not be asked to "map virtual hosts for web modules," as this carries over from the original installation.
7. Once you have finished installation, you will need to make the changes described below.

## Upgrading from EditLive! for WCM version 3.x to 4.x

The new version makes some key changes to config files and locations. You will need to make some changes when upgrading.

### Config folder location

The *ephox.config.properties* and all config xml files now reside in: */res/configs* under the EphoxEditLiveJEE ear folder on the server. Previously, they were in */res/editlivejava*.

### Role Based Config Properties File format

Previously, there was some confusion regarding how to specify role-based configurations for users/groups with spaces in their names. As such, the integration has changed so that you enter the user/group name verbatim, including any spaces or underscores.

e.g. to specify a config for the group "Content Authors", use the following line: *config.group.Content Authors=contentauthors.xml*

e.g. to specify a config for the user named "John Smith", use the following line: *config.users.John Smith=johnny.xml*

e.g. to specify a config for the group named "sys\_admins" use the following line: *config.group.sys\_admins=sysadmins.xml*

While this format isn't valid in a typical "java properties" file, it **is** valid for this config file.

### EAR file deployment

Note that this version of EditLive! for WCM is deployed as an "ear" file (Enterprise Archive). Please see the [Installation Instructions](#) for more details.

### setBackgroundMode plugin and ECM functionality

The new "Insert ECM Link" function displays an IBM-provided dialog box for inserting content from an ECM repository. To enable this functionality you need to make two changes to your config file(s):

1. Add the following line inside the `<menu name="ephox_insertmenu">` tag:

```
<menuItem name="iwwcmeclink" action="raiseEvent" value="Ephox.EditLive.WCM.insertEcmLink" />
```

2. Add the following line inside the `<plugins>` tag:

```
<plugin name="setBackgroundMode" />
```

These changes have already been applied to *sample\_eljconfig.xml*.

ECM linking is only available in IBM WCM 6.1.5 and later.

The standard/OEM version of EditLive! that is bundled with WCM has a similar menu item, but the `<menuItem>` tag is different.

# Additional Configuration Items

WCM 7+ introduces new functionality that is not enabled in EditLive! by default, due to backwards compatibility.

The configuration items for these new functions are included in the sample configuration but are disabled by default. They can be enabled by selectively removing commenting.

## Tag Helper (WCM 7+)

WCM 7.0 introduces a "Tag Helper" function, to simplify the insertion of tags within content.

To enable this functionality in EditLive!, remove the commenting around the following items in the config.xml:

```
<customMenuItem
  name="TagHelperMenuItem"
  text="Insert Tags"
  imageURL="/wps/PA_WCM_Authoring_UI/images/tagHelper.png"
  action="raiseEvent"
  value="performInsertTagIntoField" />

...

<customToolBarButton
  name="TagHelperToolBarButton"
  text="Insert Tag"
  imageURL="/wps/PA_WCM_Authoring_UI/images/tagHelper.png"
  action="raiseEvent"
  value="performInsertTagIntoField" />
```

## Export Markup (WCM 8+)

WCM 8 introduces functionality that enables the exporting of the current document in markup format.

To enable this functionality in EditLive!, remove the commenting around the following items in the config.xml:

```
<customMenuItem
  name="ExportMenuItem"
  text="Export Markup"
  imageURL="/wps/PA_WCM_Authoring_UI/images/commands/Export.png"
  action="raiseEvent"
  value="exportRTFHTML" />

...

<customToolBarButton
  name="ExportToolBarButton"
  text="Export Markup"
  imageURL="/wps/PA_WCM_Authoring_UI/images/commands/Export.png"
  action="raiseEvent"
  value="exportRTFHTML" />
```

See [Specifying Configurations in IBM WCM](#) for more information regarding customising configuration files.

# IBM Web Content Management 6.1+

- [Installing Editlive IBM WCM 6.1+](#)
- [Upgrading the Integration IBM WCM 6.1+](#)



# Installing Editlive IBM WCM 6.1+

The EditLive! WCM integration is packaged as a standard JEE EAR file. To install the EditLive! integration you can use either the IBM Integrated Solutions Console or your own installation process. The documentation below outlines how to perform an installation using the WebSphere Application Server.

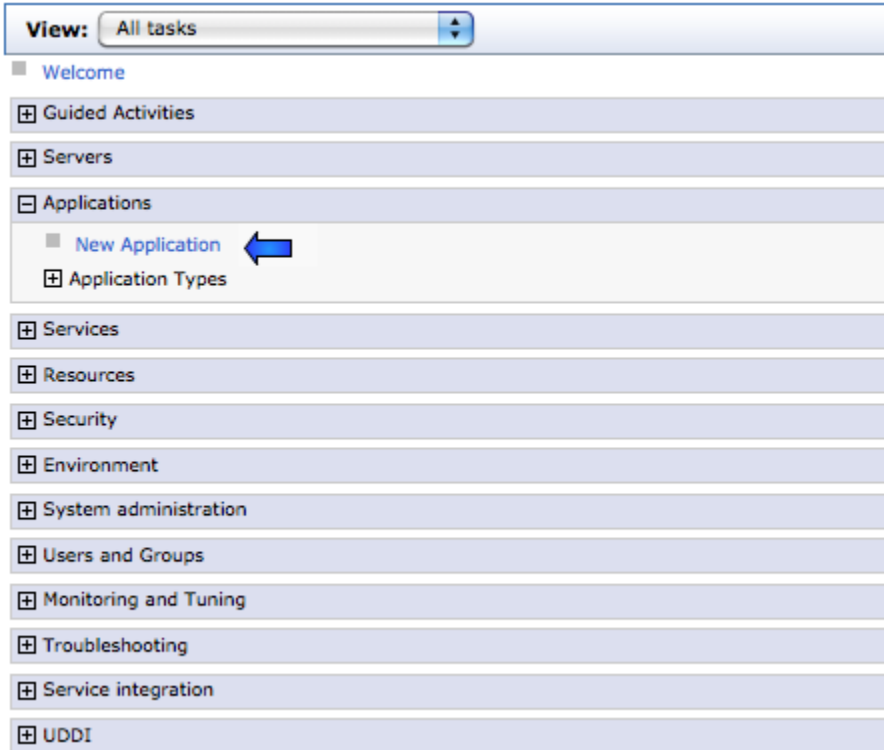
## Step 1 - Install the EAR file

The server running WebSphere\_Portal needs to be started before you begin the install process.

Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation, the url for the console will be: WebSphere Application Server 6.1 `https://[server]:9043/ibm/console_`

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to the New Application link in the console. This will involve clicking on the "Install New Application" link.



2. From the "Choose Application" form, choose the new version of the *EphoxEditLiveJEE.ear* file and click on "Next".

Enterprise Applications

Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

**Path to the new application**

Local file system

Full path  
nts\EphoxEditLiveJEE.ear

Remote file system

Full path

Context root  
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

**How do you want to install the application?**

Prompt me only when additional information is required.

Show me all installation options and parameters.

3. You now will begin the wizard for installing a new application. The EAR file has had been packaged to run well with as many default options as possible.

- In the first step the defaults are all suitable. Simply click the "Next" button.

Enterprise Applications

**Install New Application**

Specify options for installing enterprise applications and modules.

**Step 1: Select installation options**

Step 2: Map modules to servers  
Step 3: Map virtual hosts for Web modules  
Step 4: Summary

**Select installation options**

Specify the various options that are available to prepare and install your application.

Precompile JavaServer Pages files

Directory to install application

Distribute application

Use Binary Configuration

Deploy enterprise beans

Application name

Create MBeans for resources

Enable class reloading

Reload interval in seconds

Deploy Web services

Validate Input off/warn/fail

Process embedded configuration

**File Permission**

Allow all files to be read but not written to  
Allow executables to execute  
Allow HTML and image files to be read by everyone

Application Build ID

Allow dispatching includes to remote resources

Allow servicing includes from remote resources

- In step 2, you will need to ensure that the modules are deployed to the server running WebSphere Portal. Typically this will be the server called "WebSphere\_Portal". If this server does not appear in the list, it may not have been started yet. To ensure that the ear is deployed to the right location, you will need to select the server, select the module, and then click the "Apply" button.

Enterprise Applications

**Install New Application**

Specify options for installing enterprise applications and modules.

Step 1: Select installation options  
**Step 2: Map modules to servers**  
Step 3: Map virtual hosts for Web modules  
Step 4: Summary

**Map modules to servers**

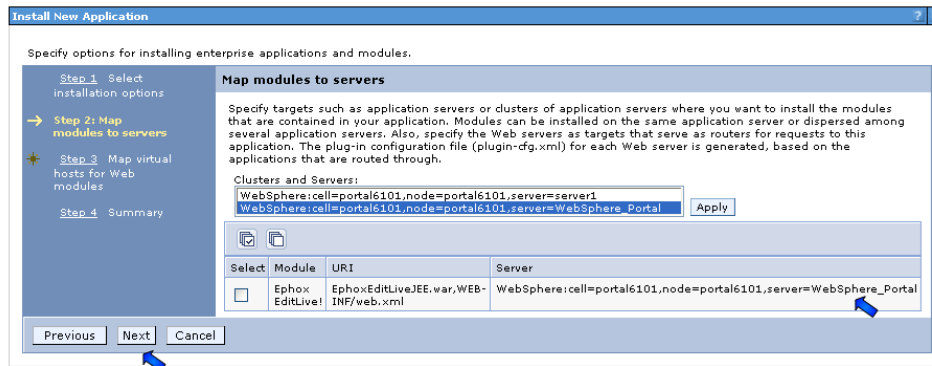
Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and Servers:

Select	Module	URI	Server
<input checked="" type="checkbox"/>	EphoxEditLive!	EphoxEditLiveJEE.war,WEB-INF/web.xml	WebSphere:cell=portal6101,node=portal6101,server=server1

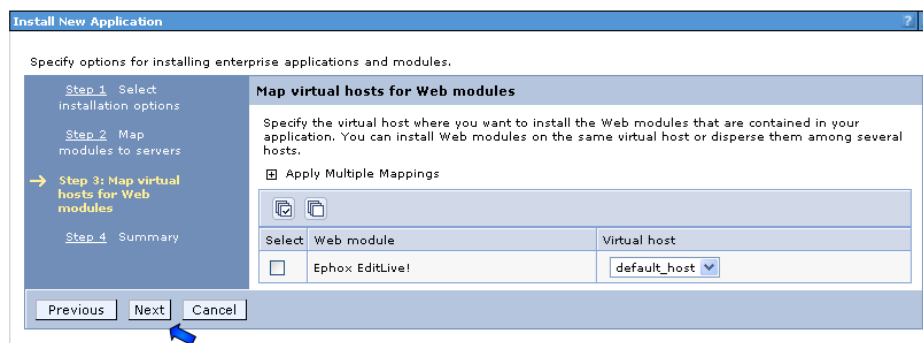
After doing this, you will expect the server section to look as indicated in the screenshot below. You can then press the "Next" button to go to step 3.

Enterprise Applications



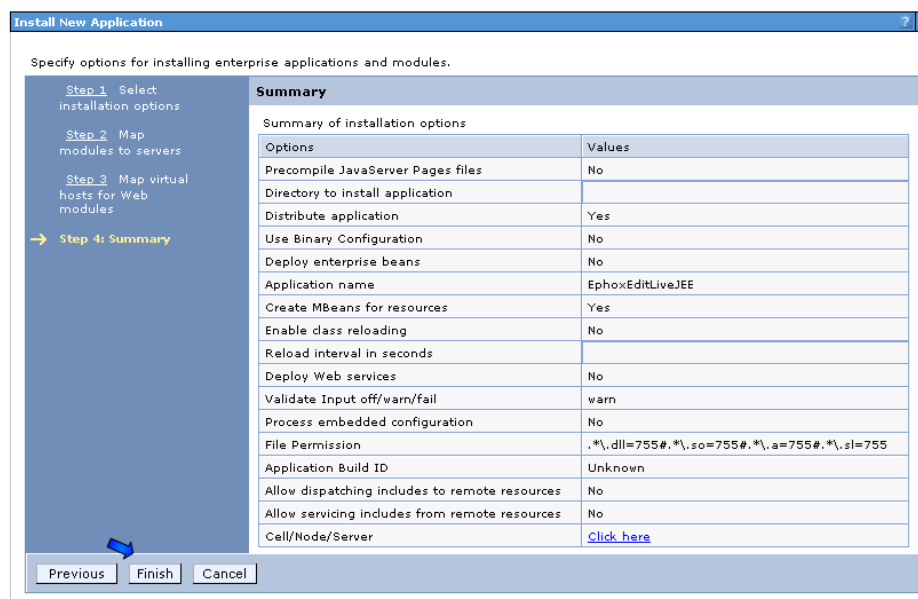
- The default options in Step 3 should be suitable, so you will typically be able to click the "Next" button and continue the wizard.

Enterprise Applications



- In step 4 you are presented with a summary. After confirming the details, click "Finish" to complete the installation.

Enterprise Applications



4. After finishing the wizard, the installation will take place, presenting the following in your browser. To ensure that the changes are applied correctly, you will need to save the configuration to the master configuration. Click the "Save" link.

**Installing...**

**If there are enterprise beans in the application, the EJB deployment process can take several minutes. Please do not save the configuration until the process completes.**

Check the SystemOut.log on the Deployment Manager or server where the application is deployed for specific information about the EJB deployment process as it occurs.

ADMA5016: Installation of EphoxEditLiveJEE started.

ADMA5067: Resource validation for application EphoxEditLiveJEE completed successfully.

ADMA5058: Application and module versions are validated with versions of deployment targets.

ADMA5005: The application EphoxEditLiveJEE is configured in the WebSphere Application Server repository.

ADMA5053: The library references for the installed optional package are created.

ADMA5005: The application EphoxEditLiveJEE is configured in the WebSphere Application Server repository.

ADMA5001: The application binaries are saved in Jopt/IBM/WebSphere/wp\_profile/wstemp/942217941/workspace/cells/porta6101/applications/EphoxEditLiveJEE.ear/EphoxEditLiveJEE.ear

ADMA5005: The application EphoxEditLiveJEE is configured in the WebSphere Application Server repository.

SECJ0400: Successfully updated the application EphoxEditLiveJEE with the appContextIDForSecurity information.

ADMA5011: The cleanup of the temp directory for application EphoxEditLiveJEE is complete.

ADMA5013: Application EphoxEditLiveJEE installed successfully.

Application EphoxEditLiveJEE installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:

- [Save](#) directly to the master configuration.
- [Review](#) changes before saving or discarding.

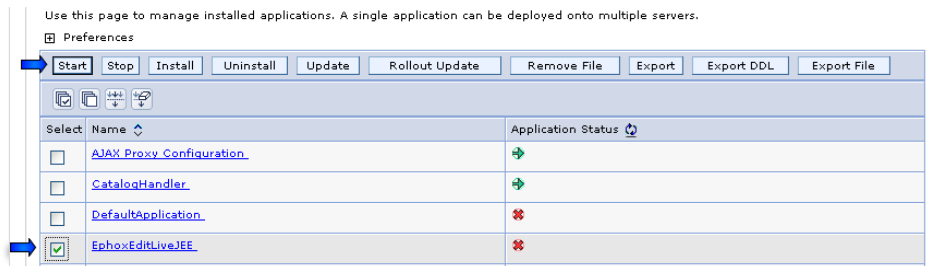
To work with installed applications, click the "Manage Applications" button.

[Manage Applications](#)

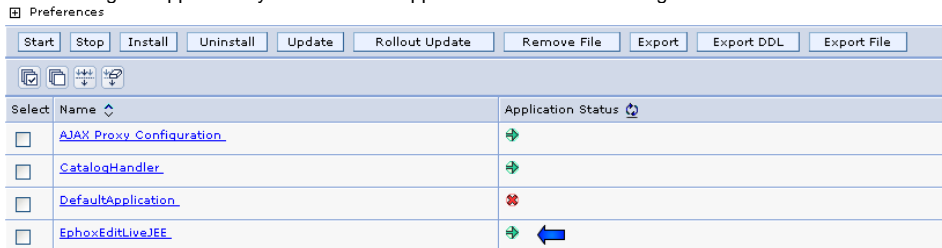
5. After installing the application, you will need to start it up. This can be done via the manage applications screen. First, navigate to the manage applications screen, clicking the WebSphere enterprise applications link.

The screenshot shows the WebSphere management console interface. At the top, there is a 'View:' dropdown menu set to 'All tasks'. Below this is a list of navigation options, each with a plus icon in a square: 'Welcome', 'Guided Activities', 'Servers', 'Applications', 'Services', 'Resources', 'Security', 'Environment', 'System administration', 'Users and Groups', 'Monitoring and Tuning', 'Troubleshooting', 'Service integration', and 'UDDI'. The 'Applications' option is expanded, showing a sub-menu with 'New Application', 'Application Types', 'Assets', and 'WebSphere enterprise applications'. A blue arrow points to the 'WebSphere enterprise applications' link. The 'Application Types' sub-menu is also expanded, showing 'WebSphere enterprise applications', 'Business-level applications', and 'Assets'.

Then select the EphoxEditLiveJEE application and click the "Start" link.

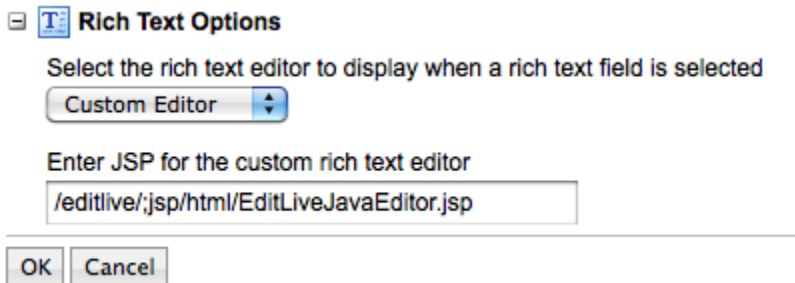


After starting the application you will see the application status now has a green arrow.



## Step 2 - Specify the EditLiveJavaEditor.jsp File in WCM

1. Log into Websphere Portal as a user with administrative WCM permissions
2. Navigate to the Web Content Management Authoring Portlet
3. Click the **Configure** link (nearby the **Help** link)
4. Expand the **Rich Text Options** section.
5. Select **Custom Editor** from the drop-down
6. In the text field displayed, enter `/editlive;/jsp/html/EditLiveJavaEditor.jsp` (The semi-colon is not a typo).



7. Click OK.

If you haven't clicked the Configure link for WCM before, this may cause the current default Library associated with the WCM portlet to be removed. To reassign the default Library assign to the WCM portlet, once inside the configure page expand the Library tab, select Web Content, and click the Add button.

## Client-Side Requirements

The EditLive! [Client-Side Requirements](#) can be found in the [Install Guide](#).

Although EditLive! itself supports a variety of JRE versions, WCM itself only supports a subset of the available JRE versions.

Visit the [WCM 6.1 Information Center](#) for an up to date list of the JRE versions supported by WCM.

# Upgrading the Integration IBM WCM 6.1+

## General Upgrade Procedure

To upgrade to a new release of the Tiny integration into WCM, perform the following steps:

- If you have made changes to *sample\_eljconfig.xml* or *customFunctionality.jsp* in order to use role based configurations, you'll need to back up these files by copying them to a location outside of your EditLive! installation.
- Follow the first step of the [Installation Instructions](#) (extract the zip file).
- Merge any changes you made from *sample\_eljconfig.xml* or *customFunctionality.jsp* into the copy of those files in the newly extracted folder.

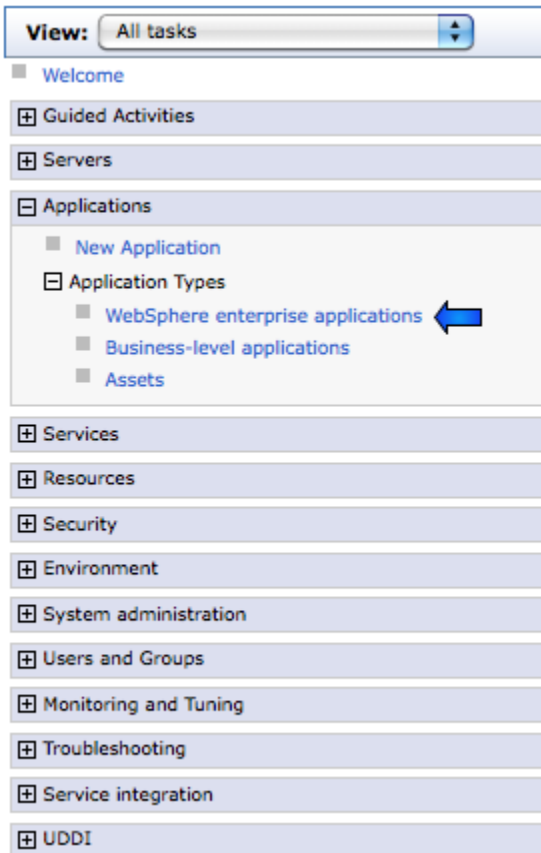
## Step 1 - Update the EAR file

The server running WebSphere\_Portal needs to be started before you begin the update process.

Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation the urls for the console will be: WebSphere Application Server 6.1 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to Enterprise Applications in the console. To do this, click on WebSphere enterprise applications link.



2. A list of installed applications will be displayed. Select the "EphoxEditLiveJEE" application from the list and press the "Stop" button.

**Enterprise Applications**

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Preferences

Start Stop **Install** Uninstall Update Rollout Update Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➔
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➔
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dojo_Resources</a>	➔
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➔
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	➔
<input type="checkbox"/>	<a href="#">FeedService</a>	➔
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➔
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➔
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➔
<input type="checkbox"/>	<a href="#">LWP_People</a>	➔
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➔
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➔
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➔
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➔
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➔
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➔
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➔
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➔
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➔

Page: 1 of 6 Total 114



- The page will reload and EphoxEditLiveJEE will now display an X in its "Application Status." Select this application again and press the "Update" button.

**Enterprise Applications**

Messages  
Application EphoxEditLiveJEE on server WebSphere\_Portal and node ephox stopped successfully.

**Enterprise Applications**  
Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Preferences

Start Stop Install Uninstall Update **Rollout Update** Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➔
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➔
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dolo_Resources</a>	➔
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➔
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	✖
<input type="checkbox"/>	<a href="#">FeedService</a>	➔
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➔
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➔
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➔
<input type="checkbox"/>	<a href="#">LWP_People</a>	➔
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➔
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➔
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➔
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➔
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➔
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➔
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➔
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➔
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➔

Page: 1 of 6 Total 114

- The next page will provide a list of application update options. The first of these is "Replace the entire application." Ensure that this option is selected and then specify the path to the latest version of the EditLive! ear file, which can either be on a local or remote file system.

Application to be updated:

EphoxEditLiveJEE

**Application update options**

Replace the entire application  
Upload an enterprise archive (\*.ear) to replace the entire installed application.

**Specify the path to the replacement ear file.**

Local file system  
Full path  
/Users/dev/Desktop/Ephox\_LWCM

Remote file system  
Full path

Context root  
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

**How do you want to install the application?**

Prompt me only when additional information is required.  
 Show me all installation options and parameters.

5. After you have finished the "Replace the entire application" section, navigate to the bottom of the page and choose "Next".

The screenshot shows a dialog box with the following content:

- Radio button:  Replace, add, or delete multiple files
- Text: Use a compressed file format such as .zip or .gzip. The compressed file is unzipped into the installed application directory. If the uploaded files exist in the application with the same paths and file names, the uploaded files replace the existing files. If the uploaded files do not exist, the files are added to the application. You can remove existing files from the installed application by specifying metadata in the compressed file.
- Section: Specify the path to the compressed file.
- Radio button:  Local file system
  - Text: Full path
  - Input field: [ ]
  - Button: Browse...
- Radio button:  Remote file system
  - Text: Full path
  - Input field: [ ]
  - Button: Browse...
- Buttons: Next, Cancel

6. From this point on, installation can be completed by following Step 1 of the [Installation Instructions](#), starting from point #4. Please note that you will not be asked to "map virtual hosts for web modules," as this carries over from the original installation.
7. Once you have finished installation, you will need to make the changes described below.

## Upgrading from EditLive! for WCM version 3.x to 4.x

The new version makes some key changes to config files and locations. You will need to make some changes when upgrading.

### Config folder location

The *ephox.config.properties* and all config xml files now reside in: */res/configs* under the EphoxEditLiveJEE ear folder on the server. Previously, they were in */res/editlivejava*.

### Role Based Config Properties File format

Previously, there was some confusion regarding how to specify role-based configurations for users/groups with spaces in their names. As such, the integration has changed so that you enter the user/group name verbatim, including any spaces or underscores.

e.g. to specify a config for the group "Content Authors", use the following line: *config.group.Content Authors=contentauthors.xml*

e.g. to specify a config for the user named "John Smith", use the following line: *config.users.John Smith=johnny.xml*

e.g. to specify a config for the group named "sys\_admins" use the following line: *config.group.sys\_admins=sysadmins.xml*

While this format isn't valid in a typical "java properties" file, it **is** valid for this config file.

### EAR file deployment

Note that this version of EditLive! for WCM is deployed as an "ear" file (Enterprise Archive). Please see the [Installation Instructions](#) for more details.

### setBackgroundMode plugin and ECM functionality

The new "Insert ECM Link" function displays an IBM-provided dialog box for inserting content from an ECM repository. To enable this functionality you need to make two changes to your config file(s):

1. Add the following line inside the `<menu name="ephox_insertmenu">` tag:

```
<menuItem name="iwwcmeclink" action="raiseEvent" value="Ephox.EditLive.WCM.insertEcmLink" />
```

2. Add the following line inside the `<plugins>` tag:

```
<plugin name="setBackgroundMode" />
```

These changes have already been applied to *sample\_eljconfig.xml*.

ECM linking is only available in IBM WCM 6.1.5 and later.

The standard/OEM version of EditLive! that is bundled with WCM has a similar menu item, but the `<menuItem>` tag is different.

# IBM Web Content Management 6.1

- [Installing EditLive IBM WCM 6.1](#)
- [Upgrading the Integration IBM WCM 6.1](#)

# Installing EditLive IBM WCM 6.1

The EditLive! IWCM integration is packaged as a standard JEE EAR file. To install the EditLive! integration you can use either the IBM Integrated Solutions Console or your own installation process. The documentation below outlines how to perform an installation using the WebSphere Application Server.

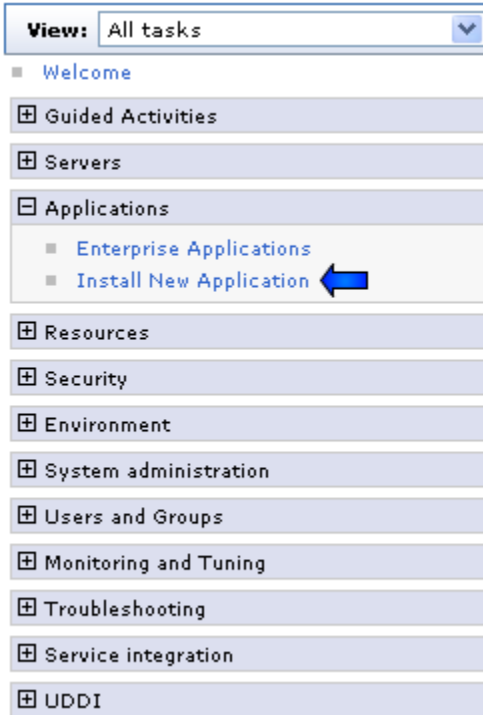
## Step 1 - Install the EAR file

The server running WebSphere\_Portal needs to be started before you begin the install process.

Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation the url for the console will be: WebSphere Application Server 6.1 <https://server:9043/ibm/console>

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to the Install New Application in the console. This will involve clicking on the "Install New Application" link as seen in Figure 1.



2. From the "Choose Application" form, choose the new version of the *EphoxEditLiveJEE.ear* file, then click on "Next".

Enterprise Applications

Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

**Path to the new application**

Local file system

Full path  
nts\EphoxEditLiveJEE.ear

Remote file system

Full path

Context root  
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

**How do you want to install the application?**

Prompt me only when additional information is required.

Show me all installation options and parameters.

3. You now will begin the wizard for installing a new application. The EAR file has had been packaged to run well with as many default options as possible.

- In the first step the defaults are all suitable. Simply click the "Next" button.

Enterprise Applications

**Install New Application**

Specify options for installing enterprise applications and modules.

**Step 1: Select installation options**

Step 2: Map modules to servers

Step 3: Map virtual hosts for Web modules

Step 4: Summary

**Select installation options**

Specify the various options that are available to prepare and install your application.

Precompile JavaServer Pages files

Directory to install application

Distribute application

Use Binary Configuration

Deploy enterprise beans

Application name

Create MBeans for resources

Enable class reloading

Reload interval in seconds

Deploy Web services

Validate Input off/warn/fail

Process embedded configuration

**File Permission**

Allow all files to be read but not written to  
 Allow executables to execute  
 Allow HTML and image files to be read by everyone

Application Build ID

Allow dispatching includes to remote resources

Allow servicing includes from remote resources

- In step 2 you will need to ensure that the modules are deployed to the server running WebSphere Portal. Typically this will be the server called "WebSphere\_Portal". If this server does not appear in the list, it may not have been started yet. To ensure that the ear is deployed to the right location, you will need to select the server, select the module, and then click the "Apply" button.

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options  
 Step 2 Map modules to servers  
 Step 3 Map virtual hosts for Web modules  
 Step 4 Summary

### Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and Servers:  
 WebSphere:cell=portal6101,node=portal6101,server=server1  
 WebSphere:cell=portal6101,node=portal6101,server=WebSphere\_Portal Apply

Select	Module	URI	Server
<input checked="" type="checkbox"/>	Ephox EditLive!	EphoxEditLiveJEE.war,WEB-INF/web.xml	WebSphere:cell=portal6101,node=portal6101,server=server1

Previous Next Cancel

After doing this, you will expect the server section to look as per the following figure. You can then press the "Next" button to go to step 3.

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options  
 Step 2 Map modules to servers  
 Step 3 Map virtual hosts for Web modules  
 Step 4 Summary

### Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and Servers:  
 WebSphere:cell=portal6101,node=portal6101,server=server1  
 WebSphere:cell=portal6101,node=portal6101,server=WebSphere\_Portal Apply

Select	Module	URI	Server
<input type="checkbox"/>	Ephox EditLive!	EphoxEditLiveJEE.war,WEB-INF/web.xml	WebSphere:cell=portal6101,node=portal6101,server=WebSphere_Portal

Previous Next Cancel

- The default options in Step 3 should be suitable, so you will typically be able to click the "Next" button and continue the wizard.

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options  
 Step 2 Map modules to servers  
 Step 3 Map virtual hosts for Web modules  
 Step 4 Summary

### Map virtual hosts for Web modules

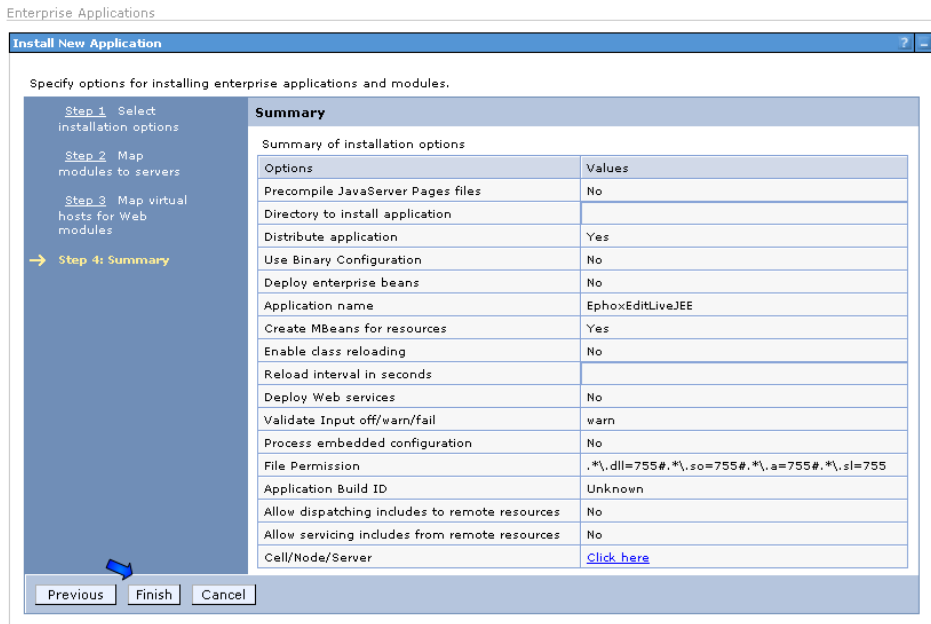
Specify the virtual host where you want to install the Web modules that are contained in your application. You can install Web modules on the same virtual host or disperse them among several hosts.

Apply Multiple Mappings

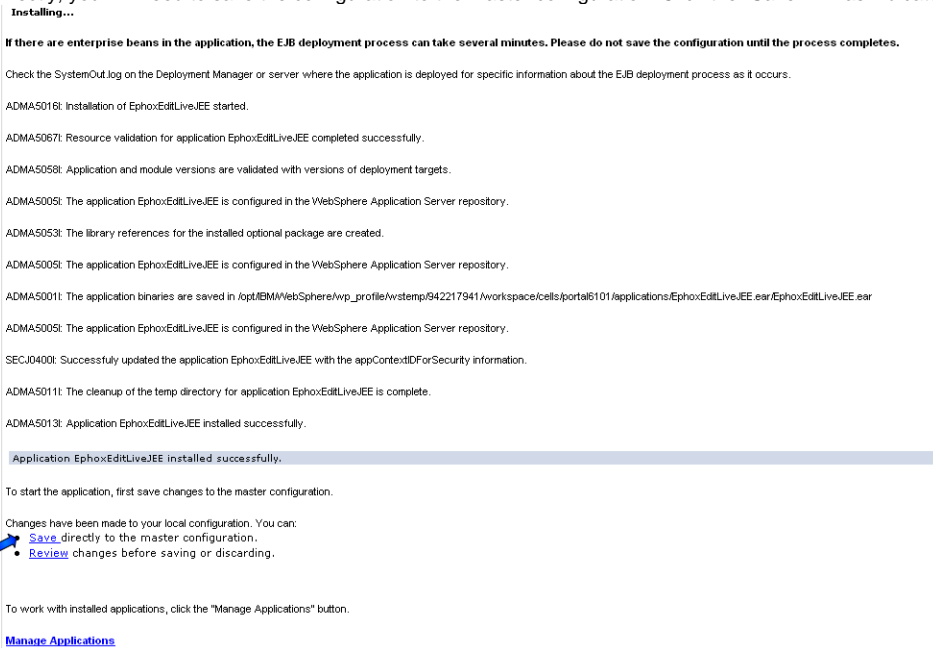
Select	Web module	Virtual host
<input type="checkbox"/>	Ephox EditLive!	default_host

Previous Next Cancel

- In step 4 you are presented with a summary. After confirming the details, click "Finish" to complete the installation.



4. After finishing the wizard, the installation will take place, presenting the following in your browser. To ensure that the changes are applied correctly, you will need to save the configuration to the master configuration. Click the "Save" link as indicated in Figure 8.

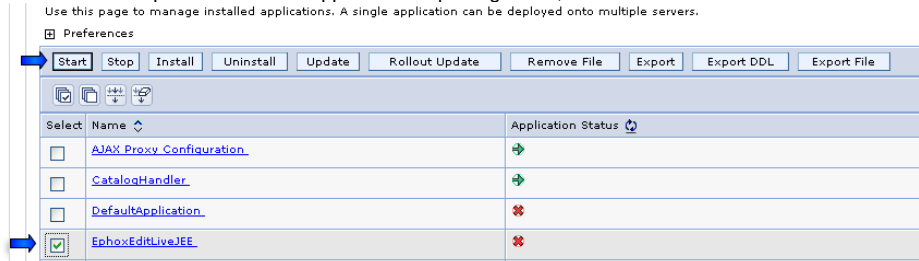


5. After installing the application you will need to start it up. This can be done via the manage applications screen. First navigate to the manage applications screen, clicking the link indicated in Figure 9.

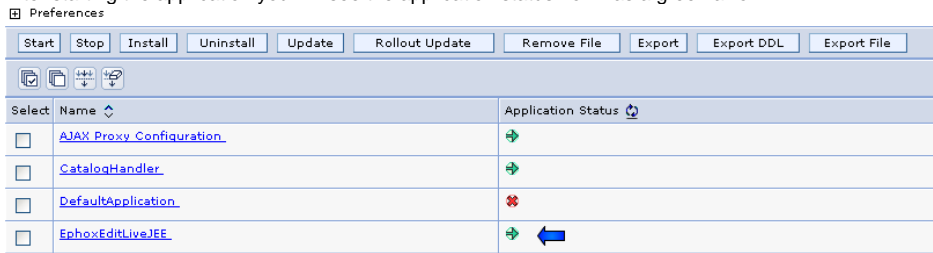




Then select the EphoxEditLiveJEE application as per Figure 10, and click the "Start" link.



After starting the application you will see the application status now has a green arrow.



## Step 2 - Specify the EditLiveJavaEditor.jsp File in WCM

1. Log into Websphere Portal as a user with administrative WCM permissions.
2. Navigate to the Web Content Management Authoring Portlet.
3. Click the Configure link (nearby the Help link).
4. Expand the Rich Text Options section.
5. Select Custom Editor from the drop-down.

6. In the text field displayed, enter /editlive;/jsp/html/EditLiveJavaEditor.jsp (The semi-colon is not a typo).

**Rich Text Options**

Select the rich text editor to display when a rich text field is selected

Custom Editor

Enter JSP for the custom rich text editor

/editlive;/jsp/html/EditLiveJavaEditor.jsp

OK Cancel

7. Click OK.

If you haven't clicked the Configure link for WCM before, this may cause the current default Library associated with the WCM portlet to be removed. To reassign the default Library assign to the WCM portlet, expand the Library tab inside the Configure page, select Web Content, and click the Add button.

## Client-Side Requirements

The EditLive! [Client-Side Requirements](#) can be found in the [Install Guide](#) of this documentation.

Although EditLive! itself supports a variety of JRE versions, WCM itself only supports a subset of the available JRE versions. Visit the [WCM 6.1 Information Center](#) for an up to date list of the JRE versions supported by WCM.

# Upgrading the Integration IBM WCM 6.1

## General Upgrade Procedure

To upgrade to a new release of the Tiny integration into WCM, perform the following steps:

- If you have made changes to *sample\_eljconfig.xml* or *customFunctionality.jsp* in order to use role based configurations, you'll need to back up these files by copying them to a location outside of your EditLive! installation.
- Follow the first step of the [Installation Instructions](#) (extract the zip file).
- Merge any changes you made from *sample\_eljconfig.xml* or *customFunctionality.jsp* into the copy of those files in the newly extracted folder.

## Step 1 - Update the EAR file

The server running WebSphere\_Portal needs to be started before you begin the update process.

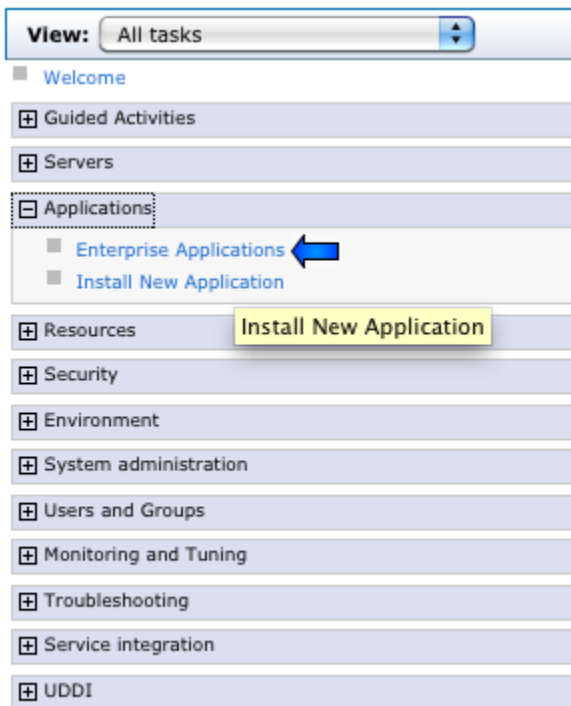
Load up your web browser and login to the WebSphere Application Server. This is a separate service running on your webserver. The port for this will vary based on your server and WebSphere Portal version. In a standard installation the urls for the console will be:

WebSphere Application Server 6.1 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

WebSphere Application Server 7.0 [https://\[server\]:9043/ibm/console](https://[server]:9043/ibm/console)

After logging in to the console, you will be able to install the application following the process below.

1. Navigate to Enterprise Applications in the console. To do this, click on the "Enterprise Applications" link.



2. A list of installed applications will be displayed. Select the "EphoxEditLiveJEE" application from the list and press the "Stop" button, as seen in Figure 2.

**Enterprise Applications**

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

☒ Preferences

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➕
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➕
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dolo_Resources</a>	➕
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➕
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	➕
<input type="checkbox"/>	<a href="#">FeedService</a>	➕
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➕
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➕
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➕
<input type="checkbox"/>	<a href="#">LWP_People</a>	➕
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➕
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➕
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➕
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➕
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➕
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➕
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➕
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➕
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➕

Page: 1 of 6    Total 114

3. The page will reload and EphoxEditLiveJEE will now display an X in its "Application Status." Select this application again and press the "Update" button, as seen in Figure 3.

**Enterprise Applications**

Messages  
Application EphoxEditLiveJEE on server WebSphere\_Portal and node ephox stopped successfully.

**Enterprise Applications**  
Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Preferences

Start Stop Install Uninstall Update **Rollout Update** Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	<a href="#">AJAX Proxy Configuration</a>	➔
<input type="checkbox"/>	<a href="#">CatalogHandler</a>	➔
<input type="checkbox"/>	<a href="#">DefaultApplication</a>	✖
<input type="checkbox"/>	<a href="#">Dolo_Resources</a>	➔
<input type="checkbox"/>	<a href="#">Enhanced_Theme</a>	➔
<input checked="" type="checkbox"/>	<a href="#">EphoxEditLiveJEE</a>	✖
<input type="checkbox"/>	<a href="#">FeedService</a>	➔
<input type="checkbox"/>	<a href="#">Feed_Servlet</a>	➔
<input type="checkbox"/>	<a href="#">IEHS_war</a>	➔
<input type="checkbox"/>	<a href="#">LWP_Mail_Servlets</a>	➔
<input type="checkbox"/>	<a href="#">LWP_People</a>	➔
<input type="checkbox"/>	<a href="#">Live_Object_Framework</a>	➔
<input type="checkbox"/>	<a href="#">MashupMaker_Integration</a>	➔
<input type="checkbox"/>	<a href="#">PA_ApplicationCatalog</a>	➔
<input type="checkbox"/>	<a href="#">PA_Banner_Ad</a>	➔
<input type="checkbox"/>	<a href="#">PA_BksFinalJSRProject</a>	➔
<input type="checkbox"/>	<a href="#">PA_Blurb</a>	➔
<input type="checkbox"/>	<a href="#">PA_Bookmarks</a>	➔
<input type="checkbox"/>	<a href="#">PA_Clients_Manager</a>	➔
<input type="checkbox"/>	<a href="#">PA_Community_Port_App</a>	➔

Page: 1 of 6 Total 114

4. The next page will provide a list of application update options. The first of these is "Replace the entire application." Ensure that this option is selected and then specify the path to the latest version of the EditLive! ear file, which can either be on a local or remote file system. The other options ("Context root" and "How do you want to install the application?") should match Figure 4 below.

Application to be updated:  
EphoxEditLiveJEE

**Application update options**

Replace the entire application  
Upload an enterprise archive (\*.ear) to replace the entire installed application.

**Specify the path to the replacement ear file.**

Local file system  
Full path:

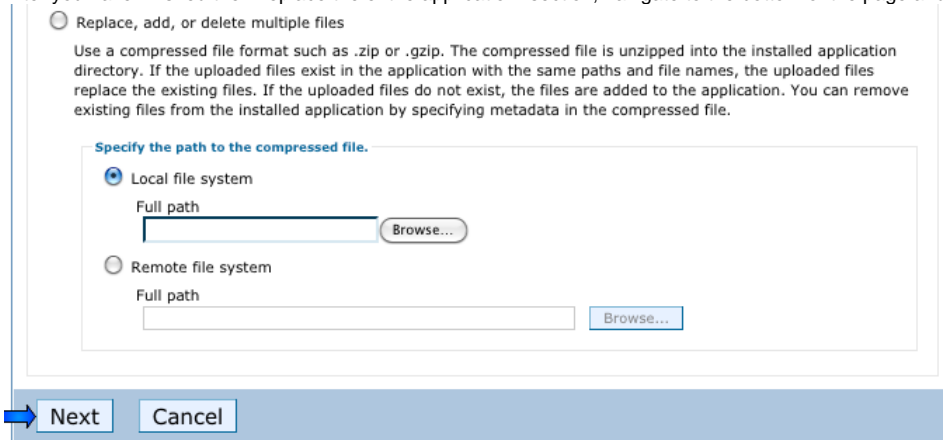
Remote file system  
Full path:

**Context root**  
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

**How do you want to install the application?**

Prompt me only when additional information is required.  
 Show me all installation options and parameters.

5. After you have finished the "Replace the entire application" section, navigate to the bottom of the page and choose "Next," as in Figure 5 below.



6. From this point on, installation can be completed by following Step 1 of the [Installation Instructions](#), starting from point #3. Please note that you will not be asked to "map virtual hosts for web modules," as this carries over from the original installation.
7. Once you have finished installation, you will need to make the changes described below.

## Upgrading from EditLive! for WCM version 3.x to 4.x

The new version makes some key changes to config files and locations. You will need to make some changes when upgrading.

### Config folder location

The *ephox.config.properties* and all config xml files now reside in: */res/configs* under the EphoxEditLiveJEE ear folder on the server. Previously, they were in */res/editlivejava*.

### Role Based Config Properties File format

Previously, there was some confusion regarding how to specify role-based configurations for users/groups with spaces in their names. As such, the integration has changed so that you enter the user/group name verbatim, including any spaces or underscores.

e.g. to specify a config for the group "Content Authors", use the following line: *config.group.Content Authors=contentauthors.xml*

e.g. to specify a config for the user named "John Smith", use the following line: *config.users.John Smith=johnny.xml*

e.g. to specify a config for the group named "sys\_admins" use the following line: *config.group.sys\_admins=sysadmins.xml*

While this format isn't valid in a typical "java properties" file, it **is** valid for this config file.

### EAR file deployment

Note that this version of EditLive! for WCM is deployed as an "ear" file (Enterprise Archive). Please see the [Installation Instructions](#) for more details.

### setBackgroundMode plugin and ECM functionality

The new "Insert ECM Link" function displays an IBM-provided dialog box for inserting content from an ECM repository. To enable this functionality you need to make two changes to your config file(s):

1. Add the following line inside the `<menu name="ephox_insertmenu">` tag:

```
<menuItem name="iwcmecmlink" action="raiseEvent" value="Ephox.EditLive.WCM.insertEcmLink" />
```

2. Add the following line inside the `<plugins>` tag:

```
<plugin name="setBackgroundMode" />
```

These changes have already been applied to *sample\_eljconfig.xml*.

ECM linking is only available in IBM WCM 6.1.5 and later.

The standard/OEM version of EditLive! that is bundled with WCM has a similar menu item, but the `<menuItem>` tag is different.

# Specifying Configurations in IBM WCM

The WCM deployment of EditLive! provides a framework for developers to specify which EditLive! configuration file is loaded into the editor depending on the current Web Content Management user's access privileges. This article describes how to create EditLive! configuration files and how to specify which files are loaded for which users.

## Custom JavaScript Integration Code

Tiny has provided a JSP file for developers where custom JavaScript methods can be specified - for example, a custom dialog called via a RaiseEvent custom toolbar button in EditLive!.

The file is *customFunctionality.jsp* and is located in the *web/jsp/html* directory of the installation package. Note that this file is updated on the server every time the integration is installed, so when upgrading the integration please ensure you keep your copy of this file.

## Creating EditLive! Configuration Files

EditLive! configuration files are XML documents which specify the appearance and functionality of an instance of the EditLive! editor. This EditLive! deployment contains the default configuration file of *sample\_eljconfig.xml*. You can modify this file and save your changes under a different file name to create your own configuration files.

Using a text editor, you can create or edit an EditLive! configuration file. The EditLive! [Developer Guide](#) packaged with this integration contains information on XML elements and attributes required in a configuration file.

The following `<menulitem>` and `<toolbarButton>` configuration file elements have been created specifically for the Tiny deployment into the Web Content Management portlet. These configuration file elements are:

- `<menulitem name="iwwcmink"/>` - A menu item allowing the user to launch the IBM dialog for creating a hyperlink to Content/Components.
- `<menulitem name="iwwcmimage"/>` - A menu item allowing the user to launch the IBM dialog for inserting images from either the user's local machine or the Component Library.
- `<menulitem name="iwwcmecmlink" action="raiseEvent" value="Ephox.EditLive.WCM.insertEcmLink"/>` - A menu item allowing the user to launch the IBM dialog for inserting content from an ECM repository.
- `<toolbarButton name="iwwcmink"/>` - A toolbar button allowing the user to launch the IBM dialog for creating a hyperlink to Content/Components.
- `<toolbarButton name="iwwcmimage"/>` - A toolbar button allowing the user to launch the IBM dialog for inserting images from either the user's local machine or the Component Library.

These configuration file elements are defined in the default configuration file (*sample\_eljconfig.xml*) packaged with this integration.

## Specifying Configuration File for EditLive! to Use (Role-based Config)

This EditLive! deployment comes packaged with an example properties file that depicts how to load a specific configuration file based on the current user's name or user group. This example properties file is called *ephox.config.properties.sample* and is located in the *res/configs* directory for your installation. The installation will be located in the folder *installedApps/EphoxEditLiveJEE.ear/EphoxEditLiveJEE.war*.

If you would like to use a configuration properties file to specify which configuration file should be used with EditLive!, change the name of this file to *ephox.config.properties*.

Each line of the configuration properties file is treated as a rule specifying which configuration file to use. EditLive! checks these rules in the following order of priority:

1. User rules (*config.user.username*)
2. Group rules (*config.group.groupname*)
3. Default rules (*config.default*)

If none of these match, then the "global" default, *sample\_eljconfig.xml*, will be used. Within each of these priorities, rules are resolved in the order they appear in the config file.

The current user's name (the combination of their first and last name for their Websphere profile) will be matched against the *config.user.FIRSTNAME LASTNAME* properties found in the file. If a match is found between the user's name and one of these properties, the corresponding configuration file is loaded into EditLive!.

There are three different ways for specifying rules: *config.user* rules, *config.group* rules, and *config.default* rules. Examples of each of these follow.

### Example - User Name:

If the current user's first name is John, second name Citizen, and *ephox.config.properties* file contains the following line:

```
config.user.John Citizen=john_config.xml
```

Then the configuration file *john\_config.xml*, located in the *res/configs* directory, will be loaded into EditLive! for this user.

The user name *can* contain spaces. Do not convert these spaces to underscores (this was required in previous versions).

### Example - User Name:

If the current user's second name is admin, no first name has been specified, and *ephox.config.properties* file contains the following line:

```
config.user.admin=admin_config.xml
```

Then the configuration file *admin\_config.xml*, located in the *res/configs* directory, will be loaded into EditLive! for this user.

For a *config.group* rule, the integration looks for a match between all the current user's Websphere group memberships and the *config.group.X* permissions.

**Example - User Role:**

If the current user belongs to the group named *lwcm\_reviewer* and *ephox.config.properties* file contains the following line:

```
config.group.lwcm_reviewer=reviewer_config.xml
```

Then the configuration file *reviewer\_config.xml*, located in the *res/configs* directory, will be loaded into EditLive! for this user.

A default configuration file can be specified in the *config.default* property.

**Example - Default:**

If the *ephox.config.properties* file only contains the following line

```
config.default=eljconfig.xml
```

Then the configuration file *eljconfig.xml*, located in the *res/configs* directory, will be loaded into for every user.

If the *ephox.config.properties* file does not contain a *config.default* property, and no *config.user* or *config.group* matches return any results, then the *sample\_eljconfig.xml* configuration file is loaded for the user.

If you don't create your own *ephox.config.properties* file, the *sample\_eljconfig.xml* configuration file will be loaded into EditLive! for any user.

If you wish to specify a different default configuration for EditLive!, Tiny encourages creating a uniquely named configuration file and specifying this under the *config.default* property in the *ephox.config.properties* file. If you simply make changes to the *sample\_eljconfig.xml* configuration file, when you upgrade the Tiny integration in the future this file will be overwritten and your changes will be lost.



# Licensing EditLive IBM WCM

In order to use Tiny EditLive!, the product will need to be registered. Upon purchasing this packaged integration, licence details will have been issued. These details are to be entered into any configuration files used with EditLive!. The default configuration files used by the integration is *sample\_eljconfig.xml*.

## Installing an EditLive! License

To install your license, repeat the following steps for any configuration file used with your integration:

1. Open the license file you received with a text editor (such as Notepad).
2. Copy the contents of the file.
3. Open your configuration file with a text editor.
4. Locate the <ephoxLicenses> section of the file.
5. After the current <license> configuration element, insert the <license> element copied from your *editlive.lic* file. You should now have two <license> configuration elements embedded in your <ephoxLicenses> element. Save the modified configuration file.

# Troubleshooting in IBM WCM

IBM OEM Support



**Please note:** If you have obtained EditLive! for IBM WCM directly from IBM you will need to lodge support requests exclusively with IBM and not Tiny. This will ensure your issues are addressed in the most expedient fashion and that you only need to work with one vendor.

## Local Multimedia Objects Don't Upload

If you've configured EditLive! to allow users to insert multimedia objects (e.g. flash, mp3), you'll notice that these files aren't uploaded from the user's local file system to the Web Content Management server. Unfortunately the IBM File Transfer Applet doesn't upload these objects to the server.

EditLive! allows developers to specify their own upload scripts to transfer images and multimedia objects to a server-side location. For more information on [configuring EditLive! for multimedia uploading](#), see the EditLive! [Developer Guide](#) packaged with this integration.

## Hyperlinks to Web Content Management Items Aren't Checked by the Broken Hyperlink Report

Due to several issues with the Web Content Management hyperlink structures, the EditLive! Broken Hyperlink Report is unable to verify if these links are valid. Any Web Content Management specific hyperlinks in EditLive!'s current document will not be displayed in the Broken Hyperlink Report, regardless of their validity.

## Logging a Support Request with Tiny

If you find a bug with either the EditLive! editor or the integration's operation itself, you can log a bug with Tiny at [Tiny Support](#).

Before logging a bug, you'll need to obtain the following details:

- Which version of EditLive! are you using (you can find this by clicking the **Ephox Icon** -> **About EditLive!** menu item)?
- Which version of the Tiny integration into WCM are you using (the integration version number can be located in the *res/editlivejava/versions.txt* file in your WCM install)?
- Which version of WCM are you running, and on what IBM Platform?

You will also need to turn HTTP debugging on for the EditLive! editor. To turn the editor debugging on, perform the following steps.

1. Open the *EditLiveJavaEditor.jsp* file located in the *jsp/html* directory for your WCM install.
2. Locate the following line of code:

```
//eljInstance.setDebugLevel("http");
```

3. Remove the // characters from the line of code.
4. Save the file.

Replicate the issue and copy the contents of the Java console to a text file. Attach this file to the support request.

Locate the *SystemErr.log* and *SystemOut.log* files (located in the *PortalServer/log* directory for your Websphere instance) and also attach these files to the support request.

If you can replicate the issue, please provide the steps required to replicate the issue, as well as any additional files required.

# Uninstalling the IBM WCM Integration

To uninstall the integration in 6.1+, perform the following steps:

1. Log into Websphere as a user with administrative WCM permissions.
2. Click Web Content -> Web Content Management.
3. Click the Configure link (next to Personalize and Help).
4. Expand the Rich Text Options tab.
5. Choose the Editor that you wish to use.
6. Click OK.

This will cause the default IBM editor to appear for all newly created items of Content. All existing items of Content edited with EditLive! will still display the EditLive! editor. This is due to unavoidable server-side functions in WCM.

To specify the default IBM editor to be used with a specific item of WCM Content (previously edited with EditLive!), perform the following:

1. Open the item of Content for editing.
2. Click the Configure link.
3. Expand the Rich Text Options tab.
4. Choose the editor that you wish to use.
5. Click OK.

# OpenText RedDot CMS Integration

The latest RedDot CMS Ephox integration is available from the Open Text Community support web site.

- [Installing EditLive for RedDot](#)
- [Licensing EditLive for RedDot](#)

# Installing EditLive for RedDot

This document assumes that you have already downloaded the latest RedDot CMS Tiny integration from the Open Text Community support web site. As of the writing of this document the downloaded file was called EditLive\_Integration\_7\_5\_1 HF1.zip

## Unzip the EditLive\_Integration\_7\_5\_1 HF1.zip file to a temporary directory. Step 1: Deploying the Files to the RedDot CMS server (V7.5 / V9 / V10)

1. Once unzipped, move the complete Ephox folder (not just the contents) from the temporary directory to C:\Program Files\RedDot\CMS\ASP\ . When done the Ephox folder is a child of the ASP folder.
2. The next step is to place EditLive into its appropriate location inside the Ephox folder. In the root directory of where you unzipped the integration you should find a " Ephox EditLive! Trial" folder. Inside that folder there is a "editlivejava.zip" file. Unzip that file - it should create a new folder called "editliveforjava".
3. In the "editliveforjava" folder navigate into the webfolder -> redistributables folder. This folder should contain a single folder named "editlivejava".
4. Move the complete editlivejava folder (not just the contents) to C:\Program Files\RedDot\CMS\ASP\Ephox
5. Move the file C:\Program Files\RedDot\CMS\ASP\Ephox\editlivejava\sample\_eljconfig.xml to C:\Program Files\RedDot\CMS\ASP\Ephox\Configuration.xml

## Step 2: Configuring RedDot to use the EditLive integration

1. In SmartTree under Start -> Administer Project Settings -> General Settings you will find "Edit Settings" in the Action Menu (right side of the screen). Click on Edit Settings.
2. Click to select the "Use external editor" checkbox
3. In the URL of external editor text field type in: Ephox (not case-sensitive - and matches the name of the folder where you placed the integration code)
4. Click OK to close the Edit Settings dialog

## Step 3: Configuring IIS Security for the Tiny Integration

1. On the CMS server, in Windows, open up Administrative Tools -> IIS Manager
2. In IIS Manager, on the left-hand tree structure, navigate to: Web Sites -> Default Web Site -> CMS -> ephox
3. Right-click on that "ephox" folder and select "Permissions"
4. In the new window, click "Add". In the window after that, click "Locations".
5. For your location, select the name of your server and not the domain name. For instance, if your CMS server is called "*ComputerName*", then you would select *ComputerName* in the list. Click "OK".
6. Back on the "Select Users and Groups" window, click "Advanced" and then click "Find Now" in the new window. You will now see a list of all your users.
7. In the list of users, select the users to give permissions to: **Administrator, Administrators, IWAM\_ *ComputerName*, IUSR\_ *ComputerName*, Reddot, ReddotUser**. After selecting all these users, click "OK". (Please note: The suffixes for IWAM and IUSR will correspond to the name of your server as discussed in step 5. Please note that the name of the "ReddotUser" account may have been changed by you during installation. If it was, you will need to select that user instead

# Licensing EditLive for RedDot

The integration will work for 30 days without a license - so if you are evaluating EditLive you can safely skip this section. However, if you have an Tiny license key you need to install that in the C:\Program Files\RedDot\CMS\ASP\Ephox\Configuration.xml file.

The license file provided by Tiny contains a small bit of XML that needs to be placed into the <licenses> section of C:\Program Files\RedDot\CMS\ASP\Ephox\Configuration.xml.

1. Open the license file provided by Tiny.
2. Copy the <license> section - make sure you get the opening and closing <license> tags but not the <ephoxLicenses> tags!
3. Open Configuration.xml in a text editor.
4. In Configuration.xml paste the <license> content into the <ephoxLicenses> section of the configuration file. Please note that there should already be a <license> tag within the file - do not remove that license section simply add this license after the existing <license>.
5. Save Configuration.xml and close the file.