Ephox EditLive! for XML Developer's Guide

Version 2.0

Ephox Corporation

Ephox EditLive! for XML Developer's Guide: Version 2.0

Ephox Corporation

Table of Contents

List of Examples

Chapter 1. Introduction

Ephox EditLive! for XML Product Information

EditLive! for XML is a forms-based authoring solution which allows for more effective and intelligent capturing of business data. The EditLive! for XML solution is based on industry standards such as XML, XML Schema and XML Stylesheets (XSLT).

The solution allows for the capture of business data as an XML file which complies with a schema generated by the Visual Designer component. The Visual Designer enables form designers to create a schema defining the data to be captured. Through the Visual Designer "views" or templates which define the form layout can be developed. EditLive! for XML combines the schema and views from the Visual Designer to allow content contributors to create an XML document which complies with the data model.

Highlights of Ephox EditLive! for XML:

- Empowers non-technical business users to create XML documents with an intuitive forms-like interface.
- A powerful Visual Designer enables non-technical users to effortlessly create templates and forms.
- Intelligent form elements streamline the process of capturing and processing business data.
- Validation based upon XML schema ensures the consistency of data
- Deployed and upgraded instantly over the Internet or intranets.
- XML open standards support enabling integration into any enterprise system.
- Cross-platform Windows, Macintosh OS X, Linux and Solaris clients.

System Requirements

This section outlines the system requirements needed to run Ephox EditLive! for XML.

Client Requirements

Microsoft Windows XP, 2000, NT 4.0, 98 or Me:

- Java 2 Platform, Standard Edition version 1.4 (installed automatically if required)
- Celeron 700-MHz or faster with at least 256MB of RAM
- Microsoft Internet Explorer 5.01 or later, Netscape Navigator 6.2 or greater, or Opera 7.0 or later.

Apple Macintosh OS X:

- Mac OS X 10.1.1 Update (includes MAC OS X Java 2 Runtime Environment)
- Safari version 1.0 or greater, or Microsoft Internet Explorer 5.13* or greater (included in MAC OS X updates).

Note

*The Visual Designer requires Java 1.4 and therefore does not support Microsoft Internet Explorer on Mac OS X. (Internet Explorer is supported on the Microsoft Windows). Apple's Safari browser is supported on Mac OS X.

Solaris 8 OE:

- Java 2 Platform, Standard Edition version 1.4 (installed automatically if required)
- Netscape Navigator 6.2 or greater

Linux:

- Java Runtime Environment version 1.4.2
- Celeron 700-MHz or faster with at least 256MB of RAM
- Netscape Navigator 7.2

EditLive! Microsoft Word Import Feature:

- Microsoft Windows XP, 2000, NT 4.0, 98 or Me
- Microsoft Office 2000 or greater installed on the client machine

Supported Application Servers

The EditLive! for XML JavaScript Edition will function on any Web server that HTML content can be served from.

Chapter 2. EditLive! for XML Install Guide

This chapter provides information on how to install EditLive! for XML.

General Server Install Instructions

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

The EditLive! for XML comes packaged as a `.zip` file or an installer package depending on the edition of the EditLive! for XML SDK to be used. To install the EditLive! for XML simply either unzip the package. Once the EditLive! for XML SDK 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. Placing this folder on a Web server will allow these portions of the SDK to be accessed.

Deploying the EditLive! for XML Sample Integrations

The sample integrations for EditLive! for XML are packaged within the folder named `webfolder`. This folder should be placed in a Web accessible directory in order to access the EditLive! for XML 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! for XML sample integrations can be accessed using a Web browser. The EditLive! for XML sample integrations are available by accessing the URL `http://`*`yourserver`*`/`*`webfolder`*`/index.html` where *`yourserver`* represents the host name of the Web server EditLive! for XML has been installed on and *`webfolder`* is the name of the virtual directory mapping to the EditLive! for XML `webfolder` directory on the file system.

EditLive! for XML Evaluation License

EditLive! for XML is packaged with a license for the `localhost` host. This is an untimed license that allows EditLive! for XML to be accessed via the `localhost` domain. Developers can use this license while evaluating and developing with EditLive! for XML. When deploying EditLive! for XML 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! for XML licenses please see the `license.txt` file packaged with EditLive! for XML. This file can be found at *`SDK_INSTALL`*`/license.txt` where *`SDK_INSTALL`* is the location that the EditLive! for XML SDK has been installed to.

Note

As EditLive! for XML is distributed with a license for the `localhost` domain it is recommended that, for evaluation and development purposes, EditLive! for XML is installed on a local Web server.

Redistributing EditLive! for XML

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

To deploy EditLive! for XML with another Web application copy the `redistributables` directory, along with its subdirectories, into the relevant Web application and ensure that the URL for the **DownloadDirectory** load-time property, the EditLive! for XML JavaScript library include and the `<spellCheck>` element `jar` attribute are correct for the Web application.

Note

The `redistributables` directory can be renamed, as can its subdirectories, without affecting the functionality of EditLive! for XML 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.

Redistributing the Java Runtime Environment from a Local Server

Developers can download and then deploy a copy of the Java Runtime Environment (JRE) from a local Web server. Redistributable JRE installers can be found on the Sun

Microsystems Java Web site [http://java.com/]. Once downloaded the JRE installer should be placed in the `redistributables/editlivexml` directory of the EditLive! for XML install or have their location specified via the JREDownloadURL property.

Packaged Image Upload Handler Scripts

Packaged with the EditLive! for XML SDK are a range of image upload handler scripts which can be used to receive images uploaded by EditLive! for XML 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! for XML's integrated image upload functionality and the packaged image upload scripts please see the documentation on Using HTTP for Image Upload in EditLive! for XML.

See Also

- Minimizing an EditLive! for XML Deployment
- Using HTTP for Image Upload in Ephox EditLive! for XML
- Licensing EditLive! for XML

Ephox EditLive! for XML JavaScript Installation Guide

Introduction

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

Note

It is recommended that you run the Ephox EditLive! for XML JavaScript SDK Install on a Web server on your local machine.

Installation Details

Other Install

Ephox EditLive! for XML SDK will be provided as a `.zip` file. The first step you need to undertake is to un-zip the downloaded file. In order to do this, simply use an un-archiving program such as WinZip [http://www.winzip.com], Power Archiver [http://www.powerarchiver.com/] or any other number of archiving utility tools available. You should un-archive the file to an appropriate location, such as `C:\editlive\`. Once you have unzipped the file, you need to set up Ephox EditLive! for XML to run on your Web server, as per the instructions below.

Web Server Deployment

To deploy the EditLive! for XML SDK on your Web server simply copy the *webfolder* 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 Ephox EditLive! for XML JavaScript Edition. Please direct your browser to the default.htm document in the root directory of your Ephox EditLive! for XML folder. For example, if you added a Web application called `editlive` to your local host Web server (on port 8080), to start viewing the documentation simply direct your Web browser to `http://localhost:8080/editlive`. If you have deployed EditLive! for XML JavaScript Edition on an external Web server then see Deploying to an External Web Server. Please work through the Ephox EditLive! for XML documentation and examples to familiarize yourselves with the product.

Ephox EditLive! for XML IIS Installation Guide

Introduction

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

Installation Details

To install Ephox EditLive! for XML on a Windows 2000 machine, running IIS 5.0:

1. Extract the `.zip` file that was downloaded for EditLive! for XML
2. Read Ephox EditLive! for XML SDK Setup Information and then click **Next**.
3. Read EditLive! for XML SDK License Information and then click **I Agree** if you accept the terms of the license. **I Agree** must be clicked if you wish to continue with the installation.
4. Select the destination directory where you would like Ephox EditLive! for XML to be installed. To change the default directory, click **Browse**. Click **Next** when you

are happy with the directory information.

5. If you are happy with the installation process thus far, click **Next** to start installing files.

6. Click **Finish** when all files have been installed.

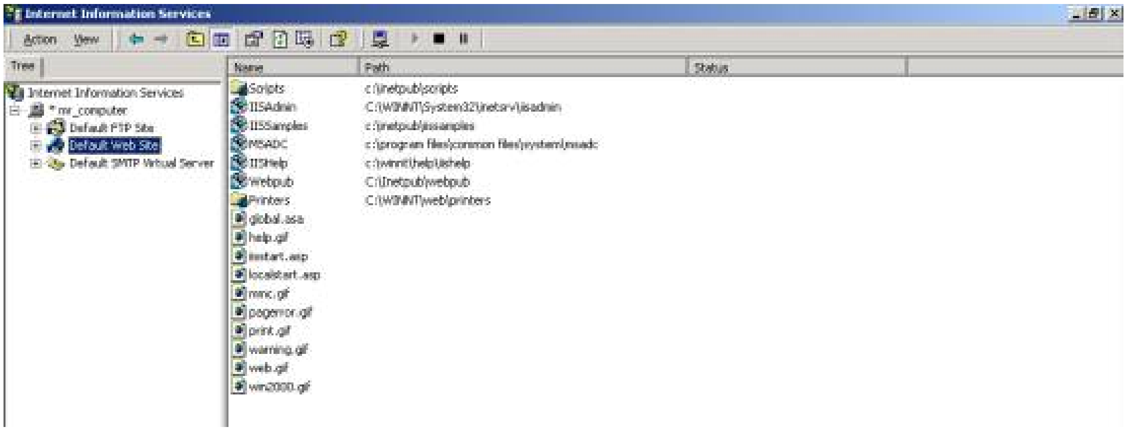
The Ephox EditLive! for XML SDK has now been installed. The next step you need to perform in order to get Ephox EditLive! for XML working on your IIS server, is to map a virtual directory for Ephox EditLive! for XML on the server.

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

1. On the **Start** menu, in **Settings** go to **Control Panel**. Now double-click on **Administrative Tools** and then **Internet Services Manager**.

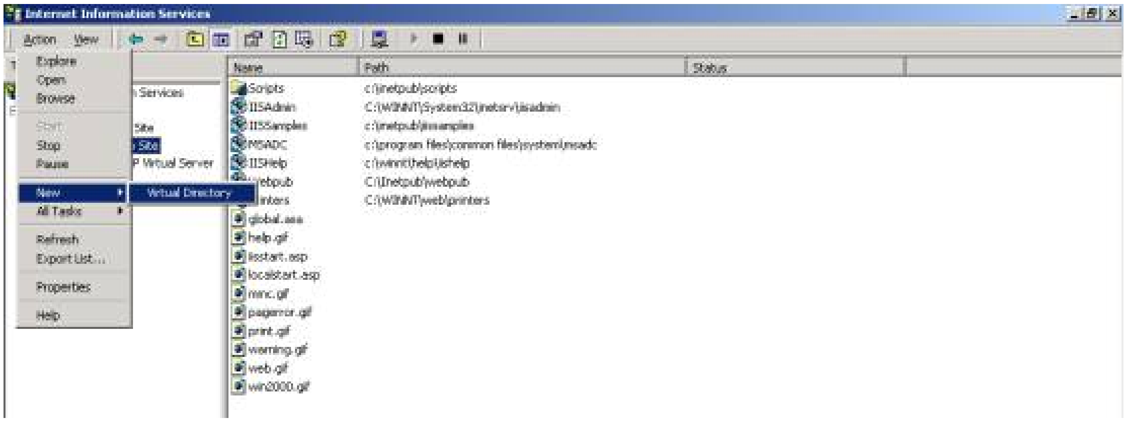
2. Expand the tree next to your Computer Name.

3. Select **Default Web Site**.


Internet Information Services
Action
View
Tree
Internet Information Services
* mr_computer
Default FTP Site
Default Web Site
Default SMTP Virtual Server

Name	Path	Status
Scripts	c:\inetpub\scripts	
IISAdmin	C:\WINNT\System32\inetsrv\iisadmin	
IISSamples	c:\inetpub\iissamples	
MSADC	c:\program files\common files\system\msadc	
IISHelp	c:\winnt\help\iishelp	
Webpub	C:\Inetpub\webpub	
Printers	C:\WINNT\web\printers	
global.asa		
help.gif		
iisstart.asp		
localstart.asp		
mmc.gif		
pagerror.gif		
print.gif		
warning.gif		
web.gif		
win2000.gif		

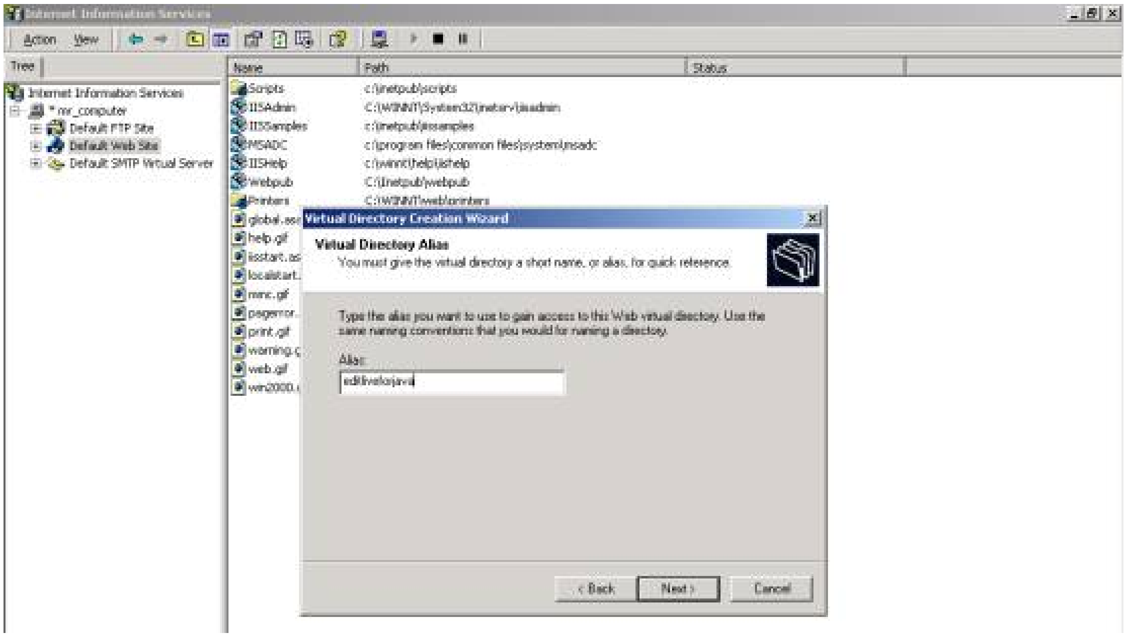
1. From the **Action** menu, select **New** > **Virtual Directory**.



Name	Path	Status
Scripts	c:\inetpub\scripts	
IISAdmin	C:\WINNT\System32\inetsrv\iisadmin	
IISSamples	c:\inetpub\iissamples	
MSADC	c:\program files\common files\system\msadc	
IISHelp	c:\winnt\help\iishelp	
Webpub	C:\Inetpub\webpub	
Printers	C:\WINNT\web\printers	
global.asa		
help.gif		
iisstart.asp		
localstart.asp		
mmc.gif		
pagerror.gif		
print.gif		
warning.gif		
web.gif		
win2000.gif		

1. Follow the **Virtual Directory Creation** Wizard.

2. In the **Alias** field, on the **Virtual Directory** Dialog, enter `editlive`


Internet Information Services
Action
View
Tree
Internet Information Services
* mr_computer
Default FTP Site
Default Web Site
Default SMTP Virtual Server
Virtual Directory Creation Wizard
< Back
Next >
Cancel

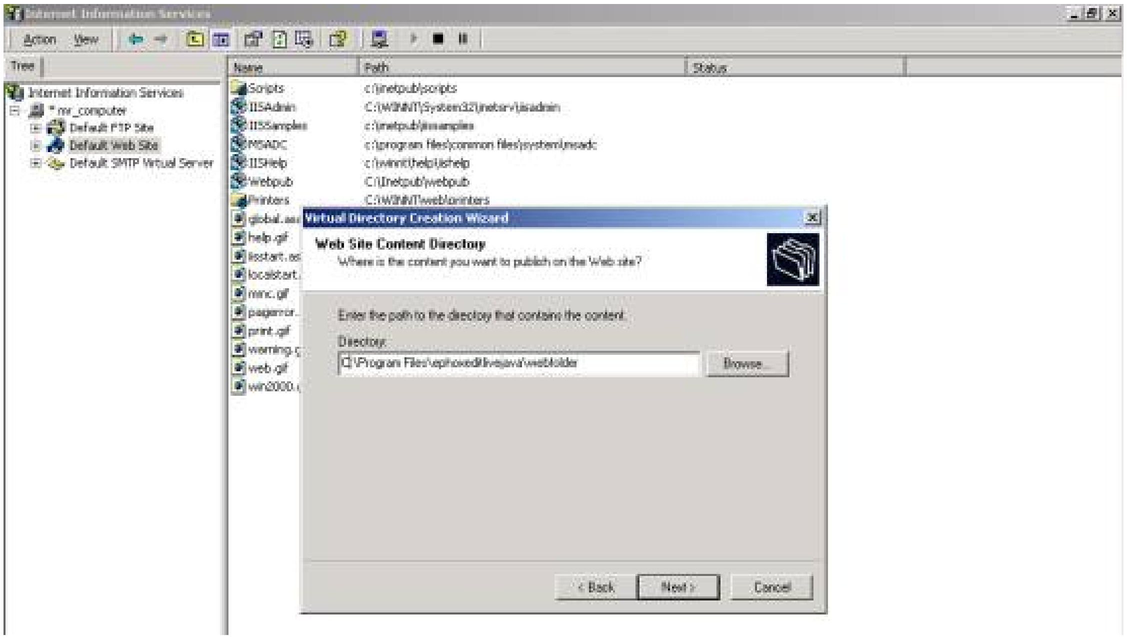
Virtual Directory Alias

You must give the virtual directory a short name, or alias, for quick reference.

Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.

Alias: editliveforjava

1. In the **Directory** field, on the **Virtual Site Content Directory** Dialog enter the correct location. e.g. "C:\Program Files\Ephox EditLive! for XML\SDK\webfolder" is the default location.



Virtual Directory Creation Wizard

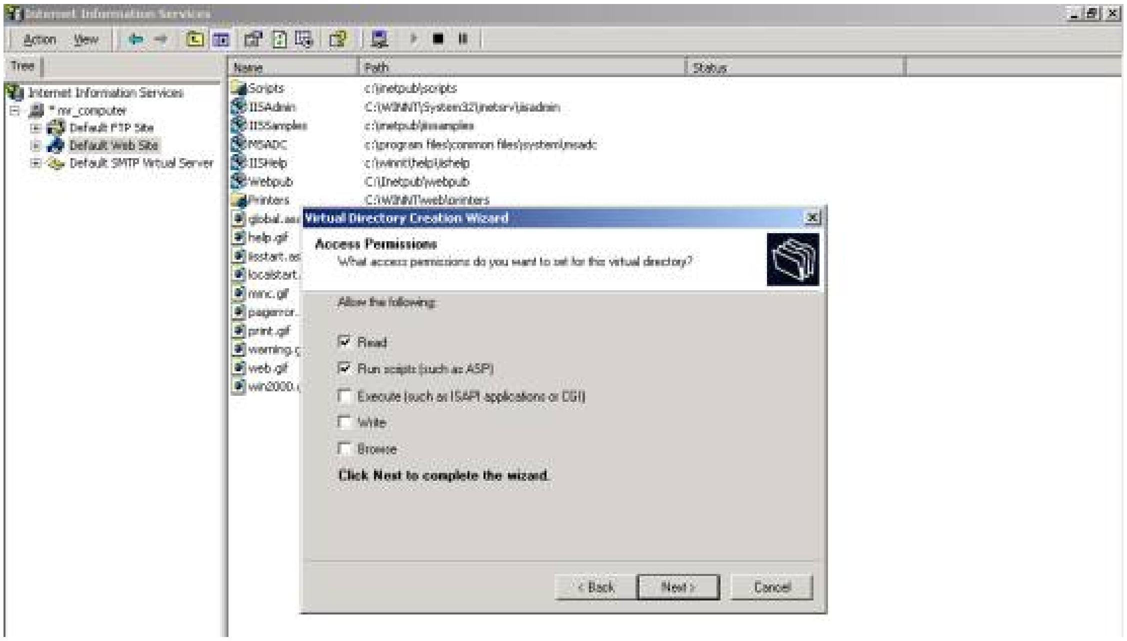
Web Site Content Directory

Where is the content you want to publish on the Web site?

Enter the path to the directory that contains the content.

Directory: C:\Program Files\ephoxeditlivejava\webfolder

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



Virtual Directory Creation Wizard

Access Permissions

What access permissions do you want to set for this virtual directory?

Allow the following:

☑ Read

☑ Run scripts (such as ASP)

☐ Execute (such as ISAPI applications or CGI)

☐ Write

☐ Browse

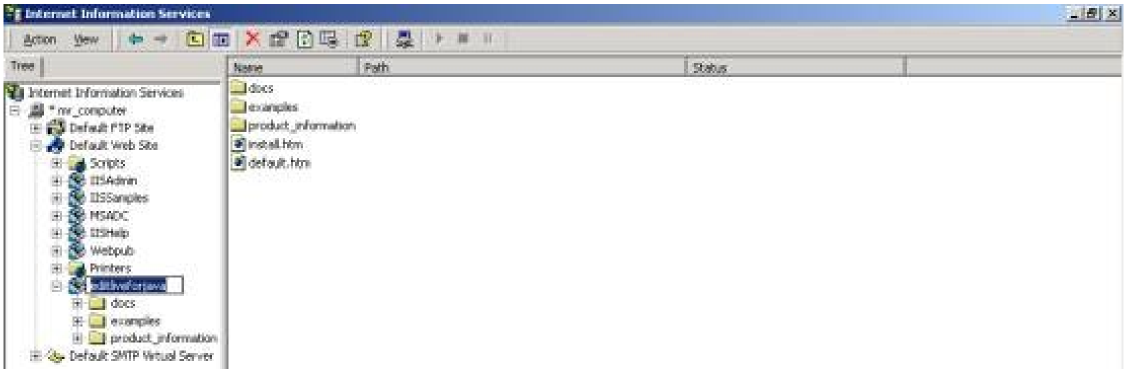
Click Next to complete the wizard.

1. Click **Next** followed by **Finish** to exit the wizard.

You will now have an `editlive` folder in the **IIS snap-in**.

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 Ephox EditLive! for XML Web component home page within the Web browser.

You have now successfully installed Ephox EditLive! for XML and customized your system correctly.

Once You Have Successfully Installed Ephox EditLive! for XML

You can now start using Ephox EditLive! for XML.

If you are using a Web server on your machine to serve the EditLive! for XML SDK then:

To look at the examples, direct your browser to: `http://localhost/`*`editliveWebfolder`* where *`editliveWebfolder`* is the name of the virtual directory in IIS.

For ease of future use, it is strongly recommended that you create a shortcut to the above link, and then add that shortcut to the **Ephox EditLive! for XML** menu.

Ephox EditLive! for XML Sample Java Server Installation with Apache Tomcat

Introduction

The aim of this article is to provide an installation guide on how to set-up Ephox

EditLive! for XML on a Tomcat [http://jakarta.apache.org/tomcat/] web server. Tomcat has been chosen as it is a freely available and simple server, that clearly shows the major steps needed to get Ephox EditLive! for XML up and running on your server of choice.

Sample Tomcat Installation Guide

Firstly you must download Tomcat if you do not already have it. You can download it from the Apache Jakarta Project [http://jakarta.apache.org/tomcat/].

Once Tomcat is installed correctly follow the below steps:

1. Stop your Tomcat web server.
2. Copy the `webfolder` directory from your Ephox EditLive! for XML installation directory (i.e. where you un-archived the downloaded Ephox EditLive! for XML ZIP file).
3. Paste the `webfolder` directory in `TOMCAT_HOME/webapps` directory. Where `TOMCAT_HOME` represents the location of your Apache Tomcat install directory.
4. Rename the `webfolder` directory to something more meaningful, such as `editlivexml`.
5.
6. Ephox EditLive! for XML should now be ready to run on your server. Simply direct your browser to `http://yourserver.com:PORT/editlivexml` (where `yourserver.com` is the name of your Web server and `PORT` is the port on the Web server that Tomcat is running on). If your web server is running on your local machine, on port 8080 for example, you would direct your browser to `http://localhost:8080/editlivexml` .

If you have followed the above steps correctly, your Tomcat web server should be now running the Ephox EditLive! for XML SDK.

Deploying to an External Web Server

Introduction

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

Note

- 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! for XML SDK.
- Ensure that you have the correct SDK for your Web server architecture.

Deployment Details

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

1. Install EditLive! for XML SDK on your local machine.
2. Locate the directory where you installed Ephox EditLive! for XML.
   - For an IIS install the default location of the EditLive! for XML SDK is: `C:\Program Files\Ephox EditLive! for XML\IIS SDK`
   - A JavaScript install does not have a default location, you will find the required files in the directory to which you unzipped the SDK.
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! for XML SDK on the external Web server direct your browser to:

   `http://`*`your_web_server`*`/`*`location_of_editlive_webfolder`*`/`

   - Where *`your_web_server`* is the name of the external Web server you have deployed to and,
   - *`location_of_editLive_webfolder`* is the location to which the EditLive! for XML SDK `webfolder` directory was copied to in the above steps

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

EditLive! for XML Client Install

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

What is Required to Use EditLive! for XML

EditLive! for XML requires that the Java Runtime Environment (JRE) version 1.4 or above is installed on the user's computer. If this is not detected then EditLive! for XML will automatically deploy and install a version of the JRE. Through the **LocalDeployment** property developers can specify where the JRE will be deployed from. This property allows developers to specify if the JRE should be deployed from the local server or downloaded over the Web from Sun Microsystems.

Installing the JRE from the Web Application Server

This is recommended when users can be expected to access EditLive! for XML via Web applications on an intranet. In this case it will be much faster to download the JRE from the local server than from Sun Microsystems' server. To deploy the JRE from the relevant application server download an installer for the appropriate version of the JRE from Sun Microsystems and place it on your Web server. Then set the **LocalDeployment** property to `true` when instantiating EditLive! for XML and also set the **JREDownloadURL** property to map to the location that the JRE is available from on your Web server .

Installing the JRE from a Sun Microsystems Server

This is recommended when users can be expected to access EditLive! for XML from a machine external to the network on which EditLive! for XML is hosted (i.e accessing EditLive! for XML over the Internet). In this case download times for the JRE can be expected to be much slower than an install over an intranet. To deploy the JRE from a Sun Microsystems' server set the **LocalDeployment** property to `false` when instantiating EditLive! for XML.

Getting the Java Runtime Environment

Copies of Sun Microsystem's Java Runtime Environment can be found on the Java Web site [http://java.com/]. 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! for XML Client

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

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

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

Dealing with a java.lang.ClassNotFound Exception

If the Java plug-in cannot locate the `editlivexml.jar` then a `java.lang.ClassNotFound` exception will be generated by the plug-in. There are two probably causes for this:

- The path set via the **DownloadDirectory** property is incorrect. Ensure that the **DownloadDirectory** property correctly provides a URL for the location of the directory containing the EditLive! for XML source, including the `editlivexml.jar` and `editlivexml.js` files.
- If it has been ensured that the path is valid for the deployment and the exception is still occurring the `editlivexml.jar` may be corrupted. Try downloading the EditLive! for XML SDK again from the Ephox Web site.

Chapter 3. Getting Started

This chapter provides information on how to get started with EditLive! for XML once it has been deployed on a system.

Licensing EditLive! for XML

Introduction

This article details how to license EditLive! for XML for use within your application. Ephox 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. For more information on licensing please contact Ephox. Licenses issued by Ephox for EditLive! for XML are bound by a licensing agreement outlined in the `license.txt` file available in the EditLive! for XML SDK. All licenses are issued at the discretion of Ephox.

Development and Trial Licenses

The EditLive! for XML 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! for XML the EditLive! for XML 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 Ephox technical support.

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

Requesting a License

The licensing mechanism of EditLive! for XML applet is reliant on the domain from which the EditLive! for XML applet is accessed. When requesting a license from Ephox please ensure that the domain name is correct for the system you are licensing EditLive! for XML for. For the purposes of licensing EditLive! for XML the domain is considered to be everything between the trailing `/` of `http://` and the next `/`, excluding the port number. For example, if the EditLive! for XML 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! for XML be required to be licensed on multiple subdomains a license which enables use on subdomains can be issued by Ephox on request. For instance, if the EditLive! for XML 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 Ephox for EditLive! for XML are bound by a licensing agreement outlined in the `license.txt` file available in the EditLive! for XML SDK. All licenses are issued at the discretion of Ephox.

Installing a License

Licenses issued by Ephox are supplied in a `.lic` file. This file will be attached to a notification email sent in response to an EditLive! for XML purchase or request for a new license. The license file contains the licensing information required by EditLive! for XML and can be opened in a text editor. The EditLive! for XML Configuration Tool can be used to install EditLive! for XML. The following steps detail how to add a license to your configuration file using a text editor.

Example 3.1. EditLive! for XML Example License

The following is an example Ephox `.lic` license file. This is the evaluation license provided with EditLive! for Java 3.0 for the `LOCALHOST` domain.

```
<ephoxLicenses>
  <license
    accountID="BB56B8DD47EF"
    activationURL="http://www.ephox.com/elregister/el2/activate.asp"
    domain="LOCALHOST"
    expiration="NEVER"
    forceActive="false"
    key="6FFF-D85E-6B15-E0A6"
    licensee="For Evaluation Only"
    product="EditLive! for Java"
    release="3.0"
    seats=""
    type="Evaluation License"
  />
</ephoxLicenses>
```

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

Example 3.2. EditLive! for XML Configuration File with Example License

The following shows an (abbreviated) EditLive! for XML configuration file which containing a license. This example is taken from the EditLive! for Java 3.0 sample configuration file.

```
<editLive>
  <document>
    ...
  </document>
  <ephoxLicenses>
    <license
      accountID="BB56B8DD47EF"
      activationURL="http://www.ephox.com/elregister/el2/activate.asp"
      domain="LOCALHOST"
      expiration="NEVER"
      forceActive="false"
      key="6FFF-D85E-6B15-E0A6"
      licensee="For Evaluation Only"
      product="EditLive! for Java"
      release="3.0"
      seats=""
      type="Evaluation License"
    />
  </ephoxLicenses>
  <htmlFilter ... />
  ...
</editLive>
```

Installing Multiple Licenses

EditLive! for XML can be used with multiple licenses. To use multiple licenses with EditLive! for XML each license should be specified in a distinct `<license>` element within the `<ephoxLicenses>` element of the EditLive! for XML configuration file. EditLive! for XML 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 and therefore multiple limited seat licenses should *not* be used in the same configuration file.

Timed Licenses

Upon request Ephox 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 2004 then the domain for the license key would appear as `WWW.YOURSERVER.COM20041030`. 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.

Using a Limited Seat License

Limited seat licenses require access to the Ephox server via the URL specified in the `activationURL` attribute. Client machines being used with a limited seat license require access to the Internet (specifically the Ephox Web page specified in the `activationURL`) on port 80 in order to register.

See Also

- `<ephoxLicenses>` element
- `<license>` element

EditLive! for XML Overview

Introduction

EditLive! for XML is a forms-based authoring solution which allows for more effective and intelligent capturing of business data. The EditLive! for XML solution is based on XML standards. The solution allows for the capture of business data as an XML file which complies with a customer defined schema which can be developed in the Visual Designer.

The Visual Designer allows form designers to create a schema defining the layout and types of data to be captured. Through the Visual Designer, templates or "views" which define form layout can be developed for the data model. EditLive! for XML combines the schema and views from the Visual Designer to allow content contributors to create an XML document which complies with the schema.

As EditLive! for XML captures content as an XML document the output of EditLive! for XML can easily be used as an input for XML-enabled enterprise systems. If required, such systems can also easily be supplied with the schema generated by the Visual Designer.

EditLive! for XML Components

The solution comprises of two in browser components; EditLive! for XML and the Visual Designer.

EditLive! for XML allows contributors to efficiently provide data through a form-based interface. The captured content is output as a XML document that complies with the supplied schema. Within EditLive! for XML a XML document can be displayed via several views which are similar to tabs in standard forms. This allows for the presentation of data to end users in various ways.

The Visual Designer allows for the design of schemas and views for use with EditLive! for XML. This allows form designers to designate the data which they wish to capture and add constraints on the data to be captured. These constraints may include the setting of data types in addition to value-range constraints. The Visual Designer also enables form designers to construct views for use in EditLive! for XML. A view controls the layout of the form as presented to users of EditLive! for XML. The Visual Designer also allows for the embedding of Intelligent Form Elements in a view, these elements include date and time pickers and action buttons which allow for elements to be easily added or removed from a form.

EditLive! for XML Life-Cycle

The EditLive! for XML solution consists of two components, the EditLive! for XML applet and the Visual Designer. To use EditLive! for XML most effectively it should be implemented such that the output of the Visual Designer is used as input for the EditLive! for XML applet. The Visual Designer provides both the schema and the views to use with an XML document loaded into EditLive! for XML.

Once the schema and views have been generated with the Visual Designer they can be used in EditLive! for XML in combination with an XML document that complies with the schema. EditLive! for XML is loaded with the schema, views and XML document allowing users to enter content. The content contributed via EditLive! for XML can then be submitted via the browser to a server for processing. As EditLive! for XML outputs an XML document the data captured by EditLive! for XML can easily be used within XML-enabled enterprise applications.

Files Used by EditLive! for XML

EditLive! for XML makes use of three types of files to effectively capture content from a user. The EditLive! for XML applet allows data to be captured as an XML document through a form constructed with the Visual Designer. Each form consists of one or more schemas and one or more views. The following provides a description of each file type, how it is used within the EditLive! for XML solution and how it is created for use with EditLive! for XML.

Data Model - XML Schema Document (XSD)	When used with EditLive! for XML an XSD provides the structure for the resulting XML document. In this way it specifies what content is captured by EditLive! for XML and also places constraints on the captured content. Such constraints can be used for form validation and ensure that a value is required, of a particular type or within a particular range of values. The Visual Designer should be used to create an XSD for use with EditLive! for XML. The Visual Designer presents form designers with a document tree structure which they can add elements to. The tree structure represents the structure of the XML document provided by the XSD. The tree structure allows designers to add elements to an EditLive! for XML form and apply constraints to elements within the data structure. The elements can then be drag-and-dropped into a view for use within EditLive! for XML. Once an element has been placed within a view it becomes accessible to content contribution interface of EditLive! for XML.
Views - XML Style Sheets (XSL)	EditLive! for XML allows multiple views to be used with a single document. Each view is presented to users as a tab in the EditLive! for XML and stored as an XML style sheet (XSL). Through the Visual Designer form designers can dictate which form fields appear on which views for a single XML document. Designers can also designate the type of intelligent form element to be used with a particular field. In conjunction with this designers can add text to a view which gives users indications on how a form is to be filled out. Designers may also add action buttons to a view in the Visual Designer. An action button is a button which is embedded within an intelligent form in EditLive! for XML and can be used to dynamically

	add or remove sections from an intelligent form.
Data - XML Document (XML)	EditLive! for XML captures content within an XML document. The XML document which is generated by EditLive! for XML complies with XSDs created with the Visual Designer. Furthermore, during the authoring process EditLive! for XML informs users if their XML document is invalid. It does this through real-time XML validation. Thus, users will be informed of any invalid content as it is entered. The XML document within EditLive! for XML is rendered using the views created within the Visual Designer. Each view is presented to users as a tab within EditLive! for XML.

Summary

EditLive! for XML provides an architecture to easily capture content via a forms based interface. Form designers can create forms for use with EditLive! for XML using the Visual Designer. Each form created for use with EditLive! for XML includes an XML schema and at least one view. The XML schema determines the structure of the resulting XML document and any data constraints. Each view specifies how the data in the XML file should be presented to users. Views can include intelligent form elements and other user interface elements which enable users to easily contribute content via EditLive! for XML.

Through EditLive! for XML users may contribute content via a forms based interface, the content is then output as an XML document. While editing the form users are presented with views of the data as created within the Visual Designer. The XML document generated by EditLive! for XML complies with the XML schema created with the Visual Designer. The XML document created by EditLive! for XML can be used as an input to other enterprise systems, the relevant XSD may also easily be supplied to these systems should it be required.

Chapter 4. Integrating EditLive! for XML

This chapter provides examples of how to integrate the EditLive! for XML solution into a Web application. The examples show how EditLive! for XML and the Visual Designer can be easily integrated within forms using JavaScript.

Basic Integrations

Introduction

This section provides information on how to create an instance of EditLive! for XML and the Visual Designer using the various editions of the EditLive! for XML SDK. Each of these examples demonstrates how to integrate EditLive! for XML in the most basic of ways so that it is running inside of a Web page.

EditLive! for XML Basic JavaScript Integration

This section of the document provides information on how to integrate EditLive! for XML into a Web page using JavaScript.

The complete source code for this example can be found in the `*INSTALL_HOME*/webfolder/examples/elxbasic/` folder where `*INSTALL_HOME*` is the location that the EditLive! for XML SDK has been installed. Also provided with this example are the view (XSL), schema (XSD) and XML documents.

Getting Started

Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic knowledge of XML, XML Schema and XSLT is recommended

Overview

In this sample EditLive! for XML is embedded into a Web page using JavaScript. In the example, EditLive! for XML is loaded with an example XSL provided by the `article.xsl` file. EditLive! for XML is also provided with the `article.xsd` file to use as an XSD. Finally, the applet is loaded with an XML document which has been URL encoded and embedded within the Web page.

This example demonstrates how to perform the following with EditLive! for XML and JavaScript:

- Embed an instance of EditLive! for XML in a Web page using JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive! for XML.
- Load a view and a data model into EditLive! for XML.
- Load a document into EditLive! for XML.

Integrating EditLive! for XML

To embed EditLive! for XML within a Web page several steps are required. Each of these steps is explained here and code samples are provided.

1. Include the `editlivexml.js` file

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

The `editlivexml.js` file contains the Ephox EditLive! JavaScript library. This library provides the interface between the browser and the EditLive! for XML `.jar` file (`editlivexml.jar`) which contains the code for the EditLive! for XML applet. The JavaScript library file can be found in the *`INSTALL_HOME`*`/redistributables/editlivexml` directory of the EditLive! for XML install.

2. Create a form to place an instance of EditLive! for XML in.

```
<form name="form1" method="POST">
```

3. Declare the EditLive! for XML JavaScript object.

```
<script language="JavaScript">
var editliveInstance;
```

4. Create a new instance of the EditLive! for XML object. When creating the EditLive! for XML 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! for XML instance with the name
// "ELApplet1", a height of 600 pixels and a width of 700 pixels.
editliveInstance = new EditLiveXML("ELApplet1", 700, 600);
```

5. Set the path to the source files for EditLive! for XML. These can be found in the `*INSTALL_HOME*/webfolder/redistributables/editlivexml` directory.

```
// This sets a relative path to the directory where
//  the EditLive! for XML redistributables can be
//  found e.g. editlivexml.jar
editliveInstance.setDownloadDirectory(
        "../../redistributables/editlivexml");
```

6. Set the URL for the EditLive! for XML configuration file.

```
// This sets a relative or absolute path to the XML
//  configuration file to use.
editliveInstance.setConfigurationFile("elconfig.xml");
```

7. Set the URL for the schema (XSD) to use with EditLive! for XML.

```
// This sets a relative or absolute path to the schema (XSD)
//  to use for XML validation.
editliveInstance.setXSDURL("article.xsd");
```

8. Set the name and the URL for the view (XSL) to use. The name used here will be used within EditLive! for XML as the label for the tab representing the `article.xsl` view.

```
// This sets a relative or absolute path to the stylesheet (XSL)
//  to use to display the XML content.
editliveInstance.addView("Article", "article.xsl");
```

9. Set the content for the applet. The content must be a valid, URL encoded XML

file. The string used in the following code is provided as an example, it is not a complete XML document. For a complete version of the source code please see the example code available with the EditLive! for XML SDK.

```
// This sets the initial content to be displayed within
//  EditLive! for XML
editliveInstance.setDocument(
       "%3C%3Fxml%20version%3D%221.0%22%20...");
```

10. Display the EditLive! for XML applet and close the `script` and `form` elements.

```
// .show is the final call and instructs the JavaScript
//  library (editlivexml.js) to insert a new EditLive! for XML
//  at the this location.
editliveInstance.show();
</script>
</form>
```

This section of code creates an instance of EditLive! for XML within the page and sets properties which affect how EditLive! for XML will be presented within the page. For more information on each of the methods here (the constructor, **setConfigurationFile**, **setDocument**, **addView**, **setXSDURL** and **show**) see the EditLive! for XML JavaScript Reference. After each of the properties have been set the **show** method is called. This method causes the instance of EditLive! for XML to be displayed in the Web page.

Example 4.1. Complete Code for Basic JavaScript Integration

The following code provides the complete code for the basic JavaScript integration of EditLive! for XML detailed in this example.

```
<html>
  <head>
    <title>Sample EditLive! for XML JavaScript Integration</title>
    <script src="../../redistributables/editlivexml/editlivexml.js">
    </script>
  </head>
  <body>
    <form name="form1" method="POST">
      <script language="JavaScript">
        var editliveInstance;
        // Create a new EditLive! for XML instance with the name
        // "ELApplet1", a height of 600 pixels and a width of 700 pixels.
        editliveInstance = new EditLiveXML("ELApplet1", 700, 600);
        // This sets a relative path to the directory where
        //  the EditLive! for XML redistributables can be
        //  found e.g. editlivexml.jar
        editliveInstance.setDownloadDirectory("../../redistributables/editlivexml");
        // This sets a relative or absolute path to the XML
        //  configuration file to use.
        editliveInstance.setConfigurationFile("elconfig.xml");
        // This sets the initial content to be displayed within
        //  EditLive! for XML
        // This sets a relative or absolute path to the schema (XSD)
        //  to use for XML validation.
        editliveInstance.setXSDURL("article.xsd");
        // This sets a relative or absolute path to the stylesheet (XSL)
        //  to use to display the XML content.
        editliveInstance.addView("Article", "article.xsl");
        // This sets the initial content to be displayed within
        //  EditLive! for XML.
        //NOTE: This document is incomplete
        editliveInstance.setDocument("%3C%3Fxml%20version%3D%221.0%22%20...");
        // .show is the final call and instructs the JavaScript
        //  library (editlivexml.js) to insert a new EditLive! for XML
        //  at the this location.
        editliveInstance.show();
      </script>
    </form>
  </body>
</html>
```

See Also

- EditLive! for XML Instantiation API Reference
- Default Values of EditLive! for XML Load-Time Properties

Visual Designer Integration

This section of the document provides information on how to integrate the Visual Designer into a Web page using JavaScript.

The complete source code for this example can be found in the

`/webfolder/examples/designerbasic/` folder where *`INSTALL_HOME`* is the location that the EditLive! for XML SDK has been installed.

Getting Started

Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic knowledge of XML Schemas and XSLT is recommended

Overview

In this sample the Visual Designer is embedded into a Web page using JavaScript. In the example, the Visual Designer is loaded with several example XSLs provided within the source of the page that the Visual Designer is embedded in. Also provided through the page is a schema document which is used as the schema within the Visual Designer.

This example demonstrates how to perform the following with the Visual Designer and JavaScript:

- Embed an instance of the Visual Designer in a Web page using JavaScript.
- Invoke methods and set parameters affecting the appearance of the Visual Designer.
- Load views and a data model into the Visual Designer.

Integrating the Visual Designer

To embed the Visual Designer within a Web page several steps are required. Each of these steps is explained here and code samples are provided.

1. Include the Ephox Visual Designer JavaScriptLibrary file, `visualdesigner.js`.

```
<script src="../../redistributables/editlivexml/visualdesigner.js">
</script>
```

The `visualdesigner.js` file contains the Ephox EditLive! JavaScript library. This library provides the interface between the browser and the Visual Designer `.jar` file (`designer.jar`) which contains the code for the Visual Designer applet. The JavaScript library file can be found in the `*INSTALL_HOME*/redistributables/editlivexml` directory of the EditLive! for XML SDK install.

2. Create a form to place an instance of the Visual Designer in.

```
<form name="form1" method="POST">
```

3. Declare the Visual Designer JavaScript object.

```
<script language="JavaScript">
var designerInstance;
```

4. Create a new instance of the Visual Designer object. When creating the Visual Designer 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 `VDApplet1`, the width of the applet is `700` pixels and the height is `600` pixels.

```
// Create a new Visual Designer instance with the name
// "VDApplet1", a height of 600 pixels and a width of 700 pixels.
designerInstance = new VisualDesigner("VDApplet1", 700, 600);
```

5. Set the path to the source files for the Visual Designer. These can be found in the `*INSTALL_HOME*/webfolder/redistributables/editlivexml` directory.

```
// This sets a relative path to the directory where
// the EditLive! for XML redistributables can be
// found e.g. designer.jar
designerInstance.setDownloadDirectory(
        "../../redistributables/editlivexml");
```

6. Set the URL for the Visual Designer configuration file.

```
// This sets a relative or absolute path to the XML
// configuration file to use.
designerInstance.setConfigurationFile("designerconfig.xml");
```

7. Set the schema (XSD) to be used with the Visual Designer.

```
// This sets the schema (XSD) to be edited with this instance of
// the Visual Designer
designerInstance.setXSDAsText("%3C%3Fxml%20version%3D%221...");
```

Note

The schema above is incomplete. When integrating the Visual Designer the XSD used with the **XSDAsText** property must be a complete XSD and URL encoded. To see the complete XSD used for this example please see the Visual Designer Basic Integration packaged with the SDK.

8. Add the views to be used within the Visual Designer. This section of code adds three views for use with the Visual Designer. The views listed here are given the names `Company`, `Second View` and `Third View`.

```
// This sets the views to be edited with this instance of the
// Visual Designer.
designerInstance.addViewAsText("Company Details",
  "%3C%3Fxml%20version%3D%221.0%22%20encoding...");
designerInstance.addViewAsText("Contact Details",
  "%3C%3Fxml%20version%3D%221.0%22%20encoding...");
designerInstance.addViewAsText("Activity Details",
  "%3C%3Fxml%20version%3D%221.0%22%20encoding...");
```

Note

The views above are incomplete. When integrating the Visual Designer the XML Style Sheets (XSL) used with the **addViewAsText** property must be complete XSLs and URL encoded. To see the complete views used for this example please see the Visual Designer Basic Integration packaged with the SDK.

9. Display the Visual Designer applet and close the `script` and `form` elements.

```
// .show is the final call and instructs the JavaScript
//  library (visualdesigner.js) to insert a new EditLive! for XML
//  at the this location.
designerInstance.show();
```

```
</script>
</form>
```

This section of code creates an instance of the Visual Designer within the page and sets properties which affect how the Visual Designer will be presented within the page. For more information on each of the methods here (the constructor, **setConfigurationFile**, **addViewAsText**, **setXSDAsText** and **show**) see the EditLive! for XML SDK JavaScript Reference. After each of the properties have been set the **show** method is called. This method causes the instance of the Visual Designer to be displayed in the Web page.

EditLive! for XML Life-Cycle Example Integration

Overview

This example demonstrates the EditLive! for XML life-cycle. The example first loads the Visual Designer in order to either create new XSLs and an XSD to use with EditLive! for XML or to edit an existing example from the EditLive! for XML sample forms package. After designing both the XSLs and XSD to use with EditLive! for XML content from the Visual Designer can be submitted to a page containing EditLive! for XML. EditLive! for XML allows for the authoring of an XML document which is presented using the XSLs and validated using the XSD designed in the Visual Designer. The XML document can then be submitted to a page which displays the document.

Required Skills

The following skills are required to work with this example:

- Basic client-side JavaScript
- Developing Web-based forms in HTML and Active Server Pages (ASP) or Java Server Pages (JSP)

Note

This example provides example code for use with both ASP (VB Script) and JSP (Java).

Example Files

The source code for this example can be found in the `webfolder/examples/lifecycle` directory. The source for the ASP scripting example is in the `asp` subdirectory and the source for the JSP scripting example is in the `jsp` subdirectory.

The each example contains the following scripting files:

- Default page - `default.asp` for ASP and `default.jsp` for JSP. This is the starting file for the example and allows for the selection of an example form to be used with EditLive! for XML. The selection made here is submitted to the designer page.
- Designer page - `designer.asp` for ASP and `designer.jsp` for JSP. The default page submits the form selection choice to this page which loads the form selection into the Visual Designer for editing. The content of the Visual Designer can then be submitted to the EditLive! for XML page.
- EditLive! for XML page - `elx.asp` for ASP and `elx.jsp` for JSP. The designer page submits the XSLs and XSDs to this page which loads the content from the Visual Designer into EditLive! for XML for XML authoring. The content of EditLive! for XML can then be submitted to the view page for output.
- View page - `view.asp` for ASP and `view.jsp` for JSP. The EditLive! for XML page submits the XML document to this page which outputs it directly to the browser.

Form Selection - Default Page

The first page of the example provides a basic interface to select an existing form or to create a new form. The exist forms used here are packaged with EditLive! for XML as an example form package. The source files for these forms can be found in the `webfolder/forms` folder of your EditLive! for XML SDK install.

1. Create the page

```
<html>
<head>
<title>EditLive! for XML - Life Cycle Example</title>
<link type="text/css" rel="stylesheet" href="stylesheet.css">
</head>

<body>
<h1>Life Cycle Example</h1>
```

2. Create a form to allow the submission of the form selection. This form submits its content to the designer page. For ASP the designer page is `designer.asp` and for JSP the designer page is `designer.jsp`.

   VBScript:

```
<form name="form1" action="designer.asp" method="GET">
```

   JSP Scripting:

```
<form name="form1" action="designer.jsp" method="GET">
```

3. Provide the selection options and finish the page.

```
<p>Select one of the example forms below to get started.<br>
  These forms are packaged with EditLive! for XML in the <i>forms</i>
  directory.</p>
  <p><select name="example">
    <option value="Contact Details">Contact Details</option>
    <option value="Expenses">Expenses</option>
    <option value="Invoice">Invoice</option>
    <option value="Leave Application">Leave Application</option>
    <option value="Partner Application">Partner Application</option>
    <option value="Quotation">Quotation</option>
    <option value="Sales Report">Sales Report</option>
    <option value="Support Issue">Support Issue</option>
    <option value="Vehicle Booking">Vehicle Booking</option>
    <option value="Create New Form">Create New Form</option>
  </select>
  <input type="submit" value="Select">
  </p>
</form>
</body>
</html>
```

Designer Page

The designer page loads the relevant example form XSLs and XSD and creates an instance of the Visual Designer for editing.

1. Start page and import any required packages.

   VBScript:

```
<%@ language="VBScript" %>
<html>
<head>
<title>EditLive! for XML - Visual Designer</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link type="text/css" rel="stylesheet" href="stylesheet.css">
</head>
<body>
    <h1>Visual Designer</h1>
    <p>Edit the form and its data structure in the Visual Designer
    and submit to EditLive! for XML.</p>
```

   JSP Scripting:

```
<%@page import="java.util.*"%>
<%@page import="java.io.*"%>
<%@page import="java.net.*"%>
<html>
<head>
<title>EditLive! for XML - Visual Designer</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link type="text/css" rel="stylesheet" href="stylesheet.css"/>
</head>
<body>
  <h1>Visual Designer</h1>
  <p>Edit the form and its data structure in the Visual Designer and
  submit to EditLive! for XML.</p>
```

2. Create a method to read data files from the Web application file system (`getFileContents`). This method accepts two parameters, the file name and the subdirectory the file is contained in.

   VBScript:

```
<%
  'This function reads a given file from the given forms subdirectory
  Function getFileContents(directory, fileName)
    Const ForReading = 1
    Dim fso, f, path, content
    'Construct the path to the file on the server
    path = Server.MapPath("../../../forms/" & directory) & "\" & fileName
    Set fso = Server.CreateObject("Scripting.FileSystemObject")
    Set f = fso.OpenTextFile(path, ForReading)
    'Read the file content
    content = f.ReadAll
    'Close and release the file
    f.Close
    Set f=Nothing
    Set fso=Nothing
    'return the file content
    getFileContents = content
  End Function
%>
```

JSP Scripting:

```
<%
  /*
  This method retrieves the content of a given file from a given subdirectory
  */
  public String getFileContents(String directory, String fileName) {
    try {
      //Construct the path to the file on the server.
      //Note that this path is resolved relative to this application's
      //root directory
      String path = getServletContext().getRealPath("forms" + File.separator
+ directory + File.separator + fileName);
      BufferedReader in = new BufferedReader(
new InputStreamReader(new FileInputStream(path)));
      StringBuffer buf = new StringBuffer();
      //Read the file into a variable one line at a time
      String line = in.readLine();
      while (line != null) {
        buf.append(line);
        buf.append("\n");
        line = in.readLine();
      }

      in.close();
      //Return the content of the file
      return buf.toString();
    } catch (Exception e) {
      e.printStackTrace();
      return "";
    }
  }
%>
```

3. Declare and initialize global variables for the page and process the request to determine which form has been selected.

   VBScript:

```
<%
  Dim xsd
  Dim views(6)
  Dim viewNames(6)
  Dim viewCount
  Dim createNew

  createNew = false

  Dim example
  If Request.QueryString("example").Count > 0 Then
    example = Request.QueryString("example").Item(1)
  End If
...
```

   JSP Scripting:

```
<%
  String xsd = "";
  List views = new ArrayList();
  List viewNames = new ArrayList();
  boolean createNew = false;

  String example = request.getParameter("example");
```

4. Read the requested data files into variables. The getFileContents method is used to retrieve the content of the requested files.

Note

An abbreviated example is shown here. For the complete series of `If-Then-Else` statements see the source code.

VBScript:

The ASP script places the content of the XSD file in the `xsd` variable and the content of each XSL forms an entry in the `views` array, finally, the label for each XSL is declared in the `viewNames` array.

```
'This series of if statements read the XSD and XSLs for the
'requested form into variables for outputting later.

'The XSD is read into the xsd variable

'Each XSL is assigned a name in the viewNames array and
'the content of the XSL is placed in the views array

If example = "Contact Details" Then
  'Read the files for the Contact Details form
  xsd = getFileContents("contactdetails", "contactdetails.xsd")

  viewNames(1) = "Company"
  views(1) = getFileContents("contactdetails", "company.xsl")
  viewNames(2) = "Contact"
  views(2) = getFileContents("contactdetails", "contact.xsl")
  viewNames(3) = "Activity"
  views(3) = getFileContents("contactdetails", "activity.xsl")
  viewCount = 3

ElseIf example = "Expenses" Then

  ...

ElseIf example = "Vehicle Booking" Then
  'Read the files for the Vehicle Booking form
  xsd = getFileContents("vehiclebooking", "vehiclebooking.xsd")

  viewCount = 1
  viewNames(1) = "Booking"
  views(1) = getFileContents("vehiclebooking", "booking.xsl")
ElseIf example = "Create New Form" Then
  'Set the create new form flag
  createNew = true
End If
```

JSP Scripting:

The JSP script places the content of the XSD file in the `xsd` string variable and the content of each XSL forms an entry in the `views` List, finally, the label for each XSL is declared in the `viewNames` list. The lists are instances of the `java.util.ArrayList` class.

```
/*
  This series of if statements read the XSD and XSLs for the
  requested form into variables for outputting later.

  The XSD is read into the xsd variable

  Each XSL is assigned a name in the viewNames list and
  the content of the XSL is placed in the views list
*/

if (example.equals("Contact Details")) {
  //Read the files for the Contact Details form
  xsd = getFileContents("contactdetails", "contactdetails.xsd");

  viewNames.add("Company");
  views.add(getFileContents("contactdetails", "company.xsl"));
  viewNames.add("Contact");
  views.add(getFileContents("contactdetails", "contact.xsl"));
  viewNames.add("Activity");
  views.add(getFileContents("contactdetails", "activity.xsl"));

}

...

else if (example.equals("Vehicle Booking")) {
  //Read the files for the Vehicle Booking form
  xsd = getFileContents("vehiclebooking", "vehiclebooking.xsd");

  viewNames.add("Booking");
  views.add(getFileContents("vehiclebooking", "booking.xsl"));
} else if (example.equals("Create New Form")) {
  //Set the create a new form flag
  createNew = true;
}
```

5. Create a form which submits content to the page with EditLive! for XML embedded in it.

   VBScript:

```
<form action="elx.asp" method="POST" name="form1">
```

JSP Scripting:

```
<form action="elx.jsp" method="POST" name="form1">
```

6. Create a form containing the Visual Designer. The Visual Designer is created and has its properties set using JavaScript.

```
<script src="../../../redistributables/editlivexml/visualdesigner.js" language="JavaScrip
</script>
<script language="JavaScript">
<!--
  var designerInstance;
  // Create a new Visual Designer instance with the name
  // "VDApplet1", a height of 700 pixels and a width of 800 pixels.
  designerInstance = new VisualDesigner("VDApplet1", 1000, 600);
  // This sets a relative path to the directory where
  // the EditLive! for XML redistributables can be
  // found e.g. designer.jar
  designerInstance.setDebugLevel("debug");
  designerInstance.setDownloadDirectory("../../../redistributables/editlivexml");
  // Set the output character set to ASCII to ensure that character
  // encoding is correct
  designerInstance.setOutputCharset("ASCII");
  // This sets a relative or absolute path to the XML
  // configuration file to use.
  designerInstance.setConfigurationFile("designerconfig.xml");
```

7. Check the `createNew` flag and load the requested XSD into the Visual Designer if the `createNew` flag has not been set. Note the use of the server-side URL encoding.

   VBScript:

```
<%
If Not createNew Then
%>
  // This sets the schema (XSD) to be edited with this instance of
  // the Visual Designer
  designerInstance.setXSDAsText("<%= Server.URLEncode(xsd) %>");
<%
End If
%>
```

JSP Scripting:

```
<%
if (!createNew){
%>
  // This sets the schema (XSD) to be edited with this instance of
  // the Visual Designer
  designerInstance.setXSDAsText("<% out.print(URLEncoder.encode(xsd)); %>");
<%
}
%>
```

8. Check the `createNew` flag and load the requested XSLs into the Visual Designer if the `createNew` flag has not been set. The XSLs and their associated names are loaded into the Visual Designer within a `for` loop. Note the use of the server-side URL encoding.

   VBScript:

```
// This sets the views to be edited with this instance of the
// Visual Designer.
<%
If Not createNew Then
  Dim i
  For i = 1 to viewCount
    Response.Write "designerInstance.addViewAsText("""
& Server.URLEncode(viewNames(i)) & """, """
& Server.URLEncode(views(i)) & """);"
  Next
End If
%>
```

   JSP Scripting:

```
// This sets the views to be edited with this instance of the
// Visual Designer.
<%
if(!createNew){
  Iterator namesIter = viewNames.iterator();
  Iterator viewsIter = views.iterator();
  while (namesIter.hasNext() && viewsIter.hasNext()) {
    out.println("designerInstance.addViewAsText(\"" +
URLEncoder.encode((String)namesIter.next()) + "\", \"" +
URLEncoder.encode((String)viewsIter.next()) + "\");");
  }
}
%>
```

9. Show the instance of the Visual Designer and finish the page. Buttons are included to allow the submission of content to the page containing EditLive! for XML and allowing navigation back to the form selection page.

```
        // .show is the final call and instructs the JavaScript
        // library (visualdesigner.js) to insert a new EditLive! for XML
        // at the this location.
        designerInstance.show();
      -->
      </script>
      <p>
        <input type="submit" value="View in ELX">
        <input type="button" value="Form Type Selection"
onclick="location.href='default.jsp';">
      </p>
    </form>
  </body>
</html>
```

EditLive! for XML Page

The EditLive! for XML page contains an instance of EditLive! for XML which is instantiated containing data from the request submitted by the designer page.

1. Import any required packages

   VBScript:

```
<%@ language="VBScript" %>
```

   JSP Scripting:

```
<%@page import="java.util.*"%>
<%@page import="java.net.*"%>
```

2. Start the HTML page

```
<html>
<head>
<title>EditLive! for XML</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link type="text/css" rel="stylesheet" href="stylesheet.css"/>
</head>
<body>
<h1>EditLive! for XML</h1>
    <p>EditLive! for XML combines the style sheet and data structure
    generated in the Visual Designer to create an electronic form
    for authoring XML.  The style sheet from the Visual Designer
    is used to present the underlying XML document while the data structure
    is used to ensure the validity of the XML document.  This page also
    allows the content of EditLive! for XML to be submitted to the server
    for viewing.</p>
```

3. Create an instance of EditLive! for XML using JavaScript.

```
<!--
** First we need to include a link to the editlivexml.js file. This file contains all of
** code required to handle loading EditLive! for XML across multiple browsers and platfor
-->
<script src="../../../redistributables/editlivexml/editlivexml.js"></script>
  <form action="view.asp" method="POST">
  <script language="JavaScript">
  <!--
  /**
   ** This section of code creates the EditLive! for XML
   ** instance and sets all of the relevant
   ** configuration information.
  */

  // Create a new EditLive! for XML instance with the name
  // "elx", a height of 650 pixels and a width of 700 pixels.
  var elx1 = new EditLiveXML("elx", 850, 600);

  // This sets a relative path to the directory where the
  // EditLive! for XML redistributables can be found e.g. editlivexml.jar
  elx1.setDownloadDirectory("../../../redistributables/editlivexml");

  // This sets a relative or absolute path to the XML configuration file to use.
  elx1.setConfigurationFile("elxconfig.xml");
```

4. Process the request variables to retrieve and set the XSD and the XSLs. Note the use of the server-side URL encoding methods.

   VBScript:

```
// This sets a relative or absolute path to the schema (XSD) to use for XML validation.
elx1.addXSDAsText("<%= Server.URLEncode(Request.Form("VDApplet1_xsd").Item(1)) %>");

// This sets a relative or absolute path to the stylesheet (XSL) to use to display the XM
<%
  Dim i
  For i = 1 to Request.Form("VDApplet1_xslt").Count
    Response.Write "elx1.addViewAsText(""" &
Server.URLEncode(Request.Form("VDApplet1_viewName").Item(i)) &
""", """ & Server.URLEncode(Request.Form("VDApplet1_xslt").Item(i)) &
""");"
  Next
%>
```

   JSP Script:

```
// This sets a relative or absolute path to the schema (XSD) to use for XML validation.
elx1.addXSDAsText("<% out.print(URLEncoder.encode(request.getParameter("VDApplet1_xsd")))

// This sets a relative or absolute path to the stylesheet (XSL)
// to use to display the XML content.
<%
  String[] names = request.getParameterValues("VDApplet1_viewName");
  String[] views = request.getParameterValues("VDApplet1_xslt");
  if (names != null && views != null) {
    for (int i = 0; i < names.length && i < views.length; i++) {
      out.println("elx1.addViewAsText(\"" +
URLEncoder.encode(names[i]) + "\", \"" +
URLEncoder.encode(views[i]) + "\");");
    }
  }
%>
```

5. Show the instance of EditLive! for XML and finish the page. Buttons are included to allow the submission of content to the page allowing the XML content to be viewed and allowing navigation back to the form selection page.

```
  // .show is the final call and instructs the JavaScript library (editlivexml.js)
  //  to insert a new EditLive! for XML
  //  at the this location.
  elx1.show();

  -->
  </script>
  <p>
  <input type="submit" value="View XML">
  <input type="button" value="Form Type Selection"
onclick="location.href='default.asp';">
  </p>
  </form>
</body>
</html>
```

View Page

This page allows the submitted content from EditLive! for XML to be viewed.

1. Import any required packages and get the submitted XML document from the request.

   VBScript:

```
<%@ language="VBScript" %>
<%
Response.Write(Request.Form("elx"))
%>
```

   JSP Scripting:

```
<%@page import="java.net.*"%>
<%
out.println(request.getParameter("elx"));
%>
```

Default Values of EditLive! for XML Load-Time Properties

The following provides a list of the deaults for the load-time properties of EditLive! for

XML SDKs.

Table 4.1. Load-Time Properties Default Values

Property Name	Default Value	Valid For
AutoSubmit	true	All SDKs
DebugLevel	info	All SDKs
LocalDeployment	false	All SDKs
ShowSystemRequirementsError	true	All SDKs
UseWebDAV	false	All SDKs

Retrieving Content from Ephox EditLive! for XML

Introduction

There are two methods with which content can be retrieved from an instance of the EditLive! for XML applet:

- POSTed as a form field with a form submit via the JavaScript **onsubmit** function (automatic submission); or
- Through the JavaScript API functions at runtime.

By default, EditLive! for XML will bind to the form **onsubmit** function.

Automatic Submission

By default the content of the EditLive! for XML applet is retrieved when a form is submitted. Because of the lack of LiveConnect support on various operating systems and browsers, EditLive! for XML 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! for XML instance and is given the name that was specified by the developer when the EditLive! for XML instance was created. For example, if ELApplet1 was specified as the name for the instance of the EditLive! for XML 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.

The Visual Designer also submits content via the JavaScript **onsubmit** function. The data model for an instance of the Visual Designer will be submitted in the form field name `designer_xsd` where `designer` is the name of the instance of the Visual Designer.

Also submitted by the Visual Designer is a collection of views. Each view is submitted within a separate form field. All the form fields containing views are given the same name, forming a collection in the request object in your application server. The form fields in which the views are submitted are named `designer_xslt` where `designer` is the name of the instance of the Visual Designer. The names corresponding to these views are contained within the collection of form fields named `designer_viewName`. The order of the collection of view names matches that of the views.

Caution

EditLive! for XML 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! for XML will not populate the hidden field with data. For this reason, please ensure you use **form.onsubmit()** to avoid this problem.

Through the use of this method the content of the EditLive! for XML applet can only be retrieved when the form in which the applet is embedded and submitted to the server. This does not facilitate the retrieval of data from the applet at runtime. To retreive the content at runtime the EditLive! for XML JavaScript API functions must be used.

Disabling the onsubmit Content Submission

The automatic submission of the content of the EditLive! for XML applet can be disabled through the use of an API setting. By setting the **AutoSubmit** property to `false` the automatic content submission can be disabled. By default the automatic submission of content by the applet is enabled.

JavaScript Runtime API Functions

In order to retrieve or manipulate the contents of the EditLive! for XML at runtime the **GetDocument** function from the JavaScript runtime API can be used. For more information on the JavaScript API please see the EditLive! for XML JavaScript API Reference. The **GetDocument** function retrieves the XML document from EditLive! for XML. This function takes a string representing the name of the JavaScript function, which receives the retrieved XML document, as a parameter. A second, optional parameter which is a boolean can also be used with this function. This second parameter can be used to indicate whether the **GetDocument** function uploads any images within the source to the Web server and adjusts their URLs accordingly.

Example

This example demonstrates how the **GetDocument** function may be used to retrieve the content of an instance of the EditLive! for XML applet called *editlive*. Note that this code is in JavaScript and it assumes that an instance of the EditLive! for XML applet called *editlive* has already been instantiated within the relevant Web page. Also shown is a JavaScript function called *JSFunctionName* which is used to set a form field with the content retrieved from the applet.

When the **GetDocument** function is called then it passes a string corresponding to the contents of the applet back to the function that is named in the parameter of the **GetDocument** function.

Note that the optional parameter forcing image upload is set to `true` to ensure any necessary images are uploaded. This ensures that the location of images correctly point to the relevant Web server within the EditLive! for XML source.

```
...
<script language="JavaScript">
  <!--
  function JSFunctionName(appletContentString){
    document.form.formField.value = appletContentString;
  }
  -->
</script>
  ...
  editlive.GetDocument('JSFunctionName',true);
  ...
```

Ensuring Output is XHTML or XML Compliant

In order to ensure that the output of content in EditLive! for XML XHTML sections is XHTML or XML compliant attributes in the htmlFilter configuration element:

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.
- It is also recommended that `encloseText` is set to `true` to ensure that content is correctly nested insides 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>`).
- It is also recommended that `encloseText` is set to `true` to ensure that content is correctly nested insides the relevant parent tags.

Summary

The content from both the EditLive! for XML and Visual Designer applets can be retrieved upon submission of the form in which the applet is embedded. The XML document content of EditLive! for XML can also be retrieved via the **GetDocument** function of the JavaScript API.

Submitting EditLive! for XML Content Directly Via HTTP POST

Introduction

Ephox EditLive! for XML can be configured to POST its content directly to a POST acceptor script on a Web server. This is useful in situations where EditLive! for XML cannot its content as part of the HTML form submission architecture. The POST operation can be called by either a custom interface item or via a JavaScript function.

Configuring the PostDocument Parameters

The PostDocument functionality of EditLive! for XML can be configured via the following parameters:

strFieldName	This parameter is required. The name of the field in the HTTP POST that EditLive! for XML uses to POST its content.

strPostURL	This parameter is required. The URL for the POST acceptor script that EditLive! for XML POSTs to.
strResponseProcessing	This parameter is required. This parameter indicates how EditLive! for XML should process the response. It has the following possible values: • `saveToDisk` • `callback` **Note** When setting this parameter to `callback` the *`strJSFunctionName`* must also be specified.
strJSFunctionName	This parameter is optional. The name of the JavaScript function to be used as a callback function. The JavaScript function specified should accept the content of the HTTP response as its only parameter. This parameter should be set in conjunction with setting the *`strResponseProcessing`* parameter to `callback`.

When using the **PostDocument** functionality as a JavaScript function then the parameters can be used as follows:

```
PostDocument(strFieldName, strPostURL, strResponseProcessing,
         [strJSFunctionName]);
```

For more information on the **PostDocument** runtime JavaScript function see the information on the **PostDocument** function in the JavaScript runtime API.

When using the **PostDocument** functionality as a custom interface item these parameters are passed through the **value** attribute of the custom item. The parameters are passed as a single string delimited by the string `##ephox##`. For more information see the example below or the article on Custom Menu and Toolbar Items for EditLive! for XML.

Example 4.2. Configuring a Custom Item to Use the PostDocument Functionality

This example demonstrates how to define a custom menu item which uses the **PostDocument** action for use within EditLive! for XML. 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>
```

Working with the HTTP Response

When EditLive! for XML POSTs its content it will receive the HTTP response. This response is processed by EditLive! for XML in accordance with the value of the *`strResponseProcessing`* parameter. When set to `saveToDisk` EditLive! for XML will open a **Save** dialog. This will allow users to save the content of the response to their local machine.

When set to `callback` EditLive! for XML will pass the content of the HTTP response to the JavaScript callback function specified by the *`strJSFunctionName`* parameter. This JavaScript method should accept the content of the response as a string, this should be the function's only parameter.

Example 4.3. Using the PostDocument Functionality with a JavaScript Callback

The following example demonstrates how to use the **PostDocument** JavaScript function with a JavaScript callback function which processes the response. The function `jsCallback` receives the content of the response from EditLive! for XML and places it in a textarea.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.js" language="JavaScript">
    </SCRIPT>
    <!--Define the JavaScript callback function for HTTP response
        processing from EditLive! for XML-->
    <SCRIPT language="JavaScript">
      function jsCallback(responseContent){
        document.exampleForm.responseContentArea.value = responseContent;
      }
    </SCRIPT>
    </HEAD>
    <BODY>
      <FORM name = exampleForm>
        <P>Click this button to POST the document in EditLive!</P>
        <P>
          <INPUT type="button" name="button1" value="Save"
onClick="editlivejs.PostDocument('editliveField',
'http://someserver/post/postacceptor.jsp', 'callback', 'jsCallback');">
        </P>
        <!--Create an instance of EditLive! for XML-->
        <SCRIPT language="JavaScript">
          <!--
          var editlivejs;
          editlivejs = new EditLiveXML("editlive",450 , 275);
          editlivejs.setDownloadDirectory("editlivexml");
          editlivejs.setLocalDeployment(false);
          editlivejs.setConfigurationFile("sample_elconfig.xml");
          editlivejs.setDocument(escape("<P>This is EditLive!</P>"));
          editlivejs.show();
        -->
      </SCRIPT>
      <P>
        <TEXTAREA name="responseContentArea" rows="20" cols="50">
        </TEXTAREA>
      </P>
    </FORM>
  </BODY>
</HTML>
```

Important Considerations when Implementing EditLive! for XML with HTTP POST

When using the EditLive! for XML applet to make the HTTP POST instead of using the browser's submit mechanisms developers must consider several things:

- If it is required that other form variables be submitted with the content of EditLive! for XML it is best to avoid using the **PostDocument** functionality of EditLive! for XML. This is because, when using the **PostDocument** functionality, the POST of EditLive! for XML 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! for XML to a completely different POST acceptor than the browser's POST. Thus, EditLive! for XML can submit its content easily to two separate processes. This can be useful when using EditLive! for XML in association with a related server side process which produces a response which should be saved to the client. For example, the POST by EditLive! for XML could send content to a server-side publishing engine which transforms the document into a PDF or some other format which is returned to the client in the HTTP response which can then be saved. In this situation the browser's POST mechanism would still be used to submit the EditLive! for XML with values from other fields for saving.
- When using the `callback` mechanism developers should be careful to provide users with adequate feedback by displaying the HTTP response in case the POST operation fails.

Summary

The **PostDocument** functionality gives developers an alternative way of submitting the content of EditLive! for XML to their systems. It can be implemented either via a custom interface item or as a JavaScript function. When implementing EditLive! for XML to POST its content directly there are several important considerations that should be addressed to ensure that EditLive! for XML will behave as expected.

See Also

- **PostDocument** function
- **Custom Menu and Toolbar Items for EditLive! for XML**

Encoding Content for Use with EditLive! for XML

Introduction

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

It is recommended that when URL encoding content that this operation be performed on the server side through the use of the appropriate server side scripting function. It is possible to use the JavaScript **escape** function however this is **not** recommended as this function does not correctly URL encode all content and may break.

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:

Example 4.4. URL Encoding with ASP

```
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:

Example 4.5. URL Encoding with ASP.NET (C#)

```
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:

Example 4.6. URL Encoding with JSP

```
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:

Example 4.7. URL Encoding with PHP

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

Note

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! for XML.

ColdFusion

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

Example 4.8. URL Encoding with ColdFusion

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

Perl

Perl does not include a URL Encode function as part of the standard libraries and therefore 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:

Example 4.9. URL Encoding with a Regular Expression in Perl

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

Summary

Content which is to be added to EditLive! for XML should be URL encoded. The methods listed above can be used to URL encode content with several popular scripting languages. The JavaScript **escape** function can also be used, however, this is not recommended as the encoding provided by this function does not comply with URL encoding.

Chapter 5. Visual Designer

Using the Visual Designer

Introduction

The Visual Designer tool enables users to create and design forms for use with EditLive! for XML. Each form consists of a schema and one or more views. Together, these files form a solution which can be developed in the Visual Designer and then deployed for use with EditLive! for XML.

Creating and Editing a Schema

The Visual Designer allows for the editing of the schema for the intelligent forms used in EditLive! for XML.

The schema provides EditLive! for XML with the rules for the data to be captured from users.

The schema created by the Visual Designer is output as an XML Schema Document (XSD) which can then be used as an input to EditLive! for XML.

Naming the Root Element

The root element is the highest level XML element that contains all other elements in your document.

By default the name of the root is `Untitled`, this should be changed to something more meaningful for the schema which is being created. For example, if the schema was for a purchase order the root element may be named `PurchaseOrder`.

To change the name of the root:

1. Right click the root element and select **Properties...** on the shortcut menu.
2. In the **Properties** dialog, enter the name in the **Name** text box.
3. Click **OK**.

Note

The names of fields cannot contain spaces and can only begin with a letter.

They can only contain alphanumeric characters, underscores ("`_`"), periods ("`.`") and hypens ("`-`").

Adding a Group

Groups contain other elements and attributes and enable the creation of controls which contain other controls such as repeating sections. For example, in a purchase order schema you might have a `Item` group which contains elements or attributes for the item such as `description`, `quantity` and `price`.

To create a new group:

1. Right click the root element and select **Add Group...** on the shortcut menu.
2. In the **Properties** dialog, enter the name of the group in the **Name** text box.
3. Click **OK**.

Adding an Element

Elements are used to store data within EditLive! for XML. Elements can exist as a child of the root element or a group. They may also be of a specific type. Creating a simple element of a specific type allows form designers to constrain the user input for a particular field. By default a simple element must appear only once within its parent. The element can be changed to be a repeating or optional element by setting the **Minimum** and **Maximum** occurrences for the element within its properties. For example, in purchase order schema may contain a `CompanyName` element for capturing the name of the company.

To create an element:

1. Right click the root element or group which is to be the parent of the element and select **Add Simple Element...** on the shortcut menu.
2. In the **Properties** dialog, enter the name of the group in the **Name** text box.
3. Click **OK**.

Adding an Attribute

Attributes are similar to elements but are less flexible and cannot be extended as easily. Like elements, attributes may also be of a specific type and have constraints placed upon their value. Attributes may occur a maximum of once within an element.

To add an attribute:

1. Right click the root element, group or element the attribute is to be added to and select **Add Attribute...** on the shortcut menu.
2. In the **Properties** dialog, enter the name of the group in the **Name** text box.
3. Click **OK**.

Specifying Data Types for Simple Elements and Attributes

The type of data which may be entered for a simple element or attribute can be specified via the Visual Designer.

anySimpleType	A field with the type `anySimpleType` can contain any content as long as it is well-formed XML. When added to a view the default control for a field of type `anySimpleType` is a text box
string	A field with the type `string` can contain any Unicode character. When added to a view the default control for this data type is a text box.
boolean	A field with the type `boolean` can contain the values `true` or `false`. When added to a view the default control for this data type is a check box.
integer	A field with the type `integer` can contain negative or positive whole number values. When added to a view the default control for this data type is a text box which will only enable a user to enter numbers or a positive (`+`) or negative (`-`) symbol. Examples of a valid integer are 1234, -4567 or +789.
decimal	A field with the type `decimal` can contain numeric values. When added to a view the default control for this data type is a text field which will only enable a user to enter numbers, a postivite (`+`) or negative (`-`) symbol, or a decimal point (`.`). Examples of a valid decimal are 0.3, -.3 or +3.5.
date	A field with the type `date` can contain date values. When added to a view the default control for this data type is a date-picker which enables users to easily enter a date manually or to select a date from a calendar. Dates will be presented to users in a format

according to their client locale.

Note

Dates are stored in the XML document in the format defined by Section 5.2.1 of ISO 8601 i.e. `CCYY-MM-DD` where `CC` represents the century, `YY` represents the year, `MM` represents the month and `DD` represents the day. Example dates include 2005-05-01 (May 1, 2005) and 1788-01-26 (January 26, 1788).

time

A field with the type `time` contains time values. When added to a view the default control for this data type is a time-picker which enables users to easily enter a time in a format specified by their client locale.

Note

Times are presented to users according to their local time zone and stored in the XML document in coordinated universal time (UTC) which is a 24-hour time as defined by Section 5.3 of ISO 8601 i.e. `hh:mm:ssZ` where `hh`, `mm` and `ss` represent hour, minute and second respectively, the `Z` indicates that times are stored in UTC. Examples of valid times are 08:33:15Z (8:33 am and 15 seconds UTC) and 21:56:25Z (9:56 pm and 25 seconds UTC).

dateTime

A field with the type `dateTime` contains both a date and a time value. When added to a view the default control for this data type is a date-time-picker which allows users to easily select both a date and a time. Date-time values will be presented to users in the format specified by their client locale.

Note

Date-times are stored in the XML document in the format defined by Section 5.4 of ISO 8601 i.e. `CCYY-MM-DDThh:mm:ssZ` where `CC` represents the century, `YY` represents the year, `MM` represents the month and `DD` represents the day. The character `T` separates the date from the time and `hh`, `mm` and `ss` represent hour, minute and second respectively. The `Z`

	indicates that the time is expressed in UTC. Times are presented to users according to their local time zone and stored as a UTC representation. An example of a valid date-time is 2004-05-01T21:56:25Z (9:56pm and 25 seconds UTC, May 1, 2004).
xhtml	A field with the type `xhtml` can contain rich text values. In an `xhtml` field authors can use formatting such as bold, italic and underline, insert images and hyperlinks and access other word processor-like functionality in EditLive! for XML. When added to a view an `xhtml` section appears as a rich text box.

Specifying Constraints and Defaults

Fields (simple elements or attributes) can have constraints placed upon their values. The types of constraints which may be placed on a field depends on the data type of the field. Each constraint consists of the property on which the constraint is applied, a comparison operator and a value. Users may select from a lists of properties and comparison operators to construct a constraint.

The schema enables designers to specify a default value for a field. When placed into EditLive! for XML the default value will appear within the control prior to any data being entered. To specify a default value for a field:

1. Right-click the simple element or attribute and select **Properties...**
2. Enter the default value in the **Default Value** text box
3. Click **OK**

Optional Elements

By default, when elements are added to the schema they have both their minimum and maximum number of occurrences set to one.

To change a simple element to be optional:

1. Right-click the simple element or attribute and select **Properties...**
2. Change the **Minimum Occurrences** to 0
3. Click **OK**

Repeating Elements

To change a simple element to be repeating:

1. Right-click the simple element or attribute and select **Properties...**
2. Enter a number greater than 1 in the **Maximum Occurrences** text box or select the check box **Unlimited Occurrences**.

When adding a repeating or optional element a button with the label `Add` *`element name`*, where *`element name`* is the name of the element, will be automatically added to the form. This button will allow users to insert extra instances of the repeating element as required.

Repeating groups may also be defined. To create a repeating group of elements:

1. Create or move the related elements within a single group.
2. Right-click the group and select **Properties...**
3. Enter a number greater than 1 in the **Maximum Occurrences** text box or select the check box **Unlimited Occurrences**.

When a repeating group is added pressing the `Add` *`element name`* button in EditLive! for XML will result in the parent element and its children being added to the document.

Moving an Element, Attribute or Group

To move an element, attribute or group:

1. Drag-and-drop the field or group to the new location.

Or to move an element, attribute or group:

1. Right-click the field you want to move and select **Move...** on the shortcut menu.
2. In the **Select Group** dialog, select a new location for the element or attribute.

Designing a View

A view is a representation of the schema which allows users to edit the underlying XML document. A view provides EditLive! for XML with a template for a form. Views contain controls, allowing the underlying XML document to be easily edited. In addition to controls views may also contain formatted text, tables, images and other rich text elements.

Creating Rich Text Content for a View

Rich text content within a view consists of all objects within a view except controls. By adding rich text elements to a view designers can provide visual cues indicating how a form should be filled out. The rich text editing capabilities of the Visual Designer are extensive and allow form designers to insert images, tables, lists, hyperlinks and formatted text into a view.

Adding Elements and Attributes to a View

There are two ways in which an element or attribute can be added to a view. Elements and attributes can be drag-and-dropped into a view, they can also be inserted at the current cursor location within the view through the **Insert at cursor** menu item.

When an element is placed within a view its attributes, children and their attributes are all placed within the view with their default renderings.

Adding Form Controls to a View

Interaction with the underlying XML document in EditLive! for XML is performed through the inclusion of form controls. These controls allow content contributors to enter data within the XML document without exposing them to the XML. Controls are embedded within a view when a field from the schema is placed within a view. Action buttons, which allow users to create and remove repeating elements can also be added.

Form Controls

Form controls provide users with a means of entering data into an underlying XML document. When an element or attribute from the schema is placed into a view the form field associated with it is the default for that data type. The default form control type for each data type is as follows:

anySimpleType	The default for this data type is a text field which will accept any character input.
string	The default for this data type is a text field which will accept any character input.

boolean	The default for this data type is a checkbox.
integer	The default for this data type is a text field which will accept only numeric characters and the + and – characters.
decimal	The default for this data type is a text field which will accept only numeric characters and the +, – and . characters.
date	The default for this data type is a date-picker.
time	The default for this data type is a time-picker.
dateTime	The default for this data type is a combined date-time-picker.
xhtml	The default for this data type is a rich text field.

There are several types of form controls which may be used for interacting with the XML document in EditLive! for XML. Form designers can specify the type of intelligent form element which is associated with a schema element or attribute via the **Control Properties** dialog. The following provides a list of available form control types and their functionality.

Check Box	A check box allows users to enter a value of either `true` or `false` into the XML document. When the check box is checked a value of `true` will be entered and when the check box is left blank a value of `false` is entered.
Date Picker	A date picker control allows users to select a date from a calendar interface. If constraints have been placed on the value of the date users may only select a date from the valid range.
Date and Time Picker	A date time picker control enables users to select both the date and the time using a calendar interface to select the date and a clock to select the time.
Drop Down List	A drop down list allows users to select a value for the field from a series of values which have been specified by the form designer. Form designers can specify a list of values via the **Control Properties** dialog when creating the view for the form. A drop down list only displays the currently selected value when it is not active. When active a drop down list will display a list of items to choose from.

List	A list allows users to select a value for the field from a series of values which have been specified by the form designer. Form designers can specify a list of values via the **Control Properties** dialog when creating the view for the form. A list differs from a drop down list in that all list options are visible to users, with the currently selected list item highlighted.	
Text Field	A text field control allows users to enter a plain text value. Valid input for a text field varies according to the data type used with the field. The following is a list of data types and the associated functionality of a text field:	
	anySimpleType or string	If the data is of the type `anySimpleType` or `string` then the text field will accept the input of any characters.
	integer	If the data is of the type `integer` then the text field will accept numeric characters and the `+` and – characters.
	decimal	If the data is of the type `decimal` then the text field will accept numeric characters and the `+`, – and . characters.
Time	A time picker control allows users to enter a time more easily. Time contols accept time in 24-hour format. A time control is split into three values, hours, minutes and seconds. These can be selected separately.	
Rich Text	Rich text controls may only be used with the XHTML data type. These give contributors access to full WYSIWYG rich text authoring. Users can include images, tables, lists, hyperlinks, formatted text and other	

rich content within these fields. Using these controls users may also copy content from Microsoft Word. Content from XHTML controls is stored within the XML document as XHTML.

Action Buttons

Action buttons are added to a view to enable users to dynamically add or remove elements. The Visual Designer will automatically add action buttons if required when an element is added to a view. There are two types of action buttons available, **Remove** and **Insert** buttons. **Remove** action buttons are associated with each instance of an element. **Insert** action buttons are associated with a group of repeating elements. The **Insert** action button for an element will be present where the next element of that kind can be inserted. For insert buttons once the maximum number of instances for the relevant element has been reached the corresponding **Insert** action button will no longer be available.

Adding Repeating Elements to a View

When adding repeating elements to a view it is important to note that a view within the Visual Designer provides a template for the element and not an exact replica of the form from EditLive! for XML. For example, if an element has its minimum occurrences attribute set to three it will appear three times within EditLive! for XML. However, in the Visual Designer it will only appear once, as a template. The template for the element, as it appears in the Visual Designer, will appear for each of the three times it is present in EditLive! for XML.

Creating Multiple Views

Multiple views can be used with a single schema. Thus designers can split a schema across several forms. It also allows a schema to be presented to the user in several different ways. Views can be added via the **Add** button on the **Views** tab within the Visual Designer. Where multiple views are present within the Visual Designer form designers can use the **Views** tab to access separate views. Double clicking on the name of a view will open the view for editing within the Visual Designer.

Visual Designer Output Formats

The Visual Designer outputs the schema and views in formats which comply with XML standards. The schema is output as an XML Schema Document (XSD) and each view is output as an XML Style Sheet (XSL). These outputs can then be used as inputs to EditLive! for XML either from a Web accessible location on the file system or embedded as text when EditLive! for XML is instantiated.

Retrieving Content from the Visual Designer

The Visual Designer submits content as part of the browser's submit mechanism. The Visual Designer submits its content when the form it is embedded in is submitted via the JavaScript **onSubmit** function. When POSTing data the Visual Designer will output the schema as an XSD in one form field. Views will be submitted as XSLs, each XSL will be placed within separate fields with the same name. This creates a collection within the POST which may be iterated over to retrieve each view.

A schema for an instance of the Visual Designer will be submitted in the form field named `designer_xsd` where `designer` is the name of the instance of the Visual Designer.

The Visual Designer submits each view in a separate field, however, each view field has the same name. Thus, when processing the browser's request the set of views from the Visual Designer can be found as a collection within the POSTed data. The form fields in which views are submitted are named `designer_xslt` where `designer` is the name of the instance of the Visual Designer.

The collection of names for the views generated in the Visual Designer are also POSTed by the Visual Designer. Each view name is submitted in a separate field, however, each view name field has the same name in the POST. Thus, when processing the browser's request the set of view names from the Visual Designer can be found as a colelction within the POSTed data. The order in which the view names are POSTed is the same as that for the views. The form fields in which view names are submitted are named `designer_viewName` where `designer` is the name of the instance of the Visual Designer.

Exporting Files from the Visual Designer

The Visual Designer enables users to export a view or schema to the file system. This enables users to store schemas and views on the file system which may be used as inputs for EditLive! for XML when placed in a Web accessible location. When exporting the schema it will be stored as an XSD on the file system with the extension `.xsd`, the XSD may be exported by using the **Export Schema** menu item. When exporting a view it will be stored as an XSL on the file system, with the extension `.xsl`, XSLs may be exported by using the **Export StyleSheet** menu item.

Conclusion

The Visual Designer is a tool which allows users to create schemas and views which are then used to construct an intelligent form within EditLive! for XML. The schema created within the Visual Designer designates what data is to be captured, the type of data and any constraints on the data. It also provides form designers with a means of logically grouping related data.

Views created within the Visual Designer allow form designers to specify how data will be presented to users. A view is a mixture of rich text objects and form controls. The rich text components of a view give designers the flexibility to add formatted text, images, tables, lists and other elements to each view while the addition of form controls provides an interface to the underlying XML document within EditLive! for XML.

The output from the Visual Designer is used within EditLive! for XML to validate and render XML documents. The Visual Designer provides two forms of output. The schema is output as an XML Schema Document (XSD) and each view is output as a separate XML Style Sheet (XSL) document. These can then be used by EditLive! for XML by reading them from a Web accessible location or directly from the page which EditLive! for XML is embedded in.

Chapter 6. Cascading Style Sheet Support

Ephox EditLive! for XML and CSS

Introduction

Ephox EditLive! for XML provides support for the use of Cascading Style Sheets (CSS) to enforce formatting standards easily by separating content from formatting. Styles can be used within XHTML sections of an EditLive! for XML document. Styles will also be used to render text within EditLive! for XML. Styles can be specified 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 [http://www.w3.org/MarkUp/Guide/Style].

Adding Styles to EditLive! for XML

EditLive! for XML provides several methods of adding style information to a document. Furthermore, EditLive! for XML recognizes style information and populates the style drop-down list box accordingly. EditLive! for XML 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.

Style information for EditLive! for XML can be provided in the following ways:

- The XML configuration file of EditLive! for XML may include style information in a couple of ways. Note that, with the exception of styles defined inline, style information specified through the use of the XML configuration file takes precedence over style information specified in any other way.
  - an external style sheet may be specified through the use of the **<link>** element.
  - an embedded style sheet can be specified through the **<style>** element.
- Microsoft Word Styles - Styles can be imported from a Microsoft Word document.

It should be noted that EditLive! for XML's 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 is any conflict between style sheets.

Specifying Inline and Block Styles

The way in which styles are specified within a style sheet affects the way they can be applied within EditLive! for XML. 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 XML. A block style which defines that paragraph text should be blue can be defined as follows:

```
p.blue{color: blue}
```

Inline styles in EditLive! for XML are applied using the `<SPAN>` XHTML element. Thus, to define a style which can only be used inline it should be defined as a class to be used with the `<SPAN>` tag. These styles are designated by a a mark on the styles drop down in EditLive! for XML. An inline style which would specify text that is to have a red color would be as follows:

```
span.redtext{color:red}
```

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 XML styles drop down, once marked with the ¶ symbol and again with a **a** mark. 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}
```

Including Styles via the Configuration File

Linking to an External Style Sheet

EditLive! for XML can be configured to use external style sheets by specifying a value with the **<link>** element in the configuration file.

When EditLive! for XML links to multiple style sheets, the last style sheet added will have priority if there is a conflict with an earlier style sheet.

For example, if stylesheet1.css defined H1 as:

```
H1 { font-family: Arial,Helvetica,sans-serif; font-size: 24pt; }
```

and another style sheet linked **after** stylesheet1.css 1, defined H1 as:

```
H1 { font-family: Arial,Helvetica,sans-serif; font-size: 48pt; }
```

the value of stylesheet1.css for H1 would be overridden by the value given in the second style sheet (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

Through the **<style>** element of the EditLive! for XML configuration file an embedded style sheet can specified. Styles listed in a style sheet embedded via the XML configuration file take precedence over styles otherwise defined. The following would configure EditLive! for XML 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

Unless the **styleOption** attribute of the **<wordImport>** element in the configuration file is set to `clean`, styles may be imported from Microsoft Word. When styles are imported in this manner they are added to the embedded style sheet of the document in the Visual Designer. If a style of the same class already exists in the Visual Designer it will take precedence over any style from Microsoft Word.

Note

Style information cannot be imported from Microsoft Word when using EditLive! for XML. It is recommended that the **styleOption** attribute of the **<wordImport>** element in the configuration file is set to `clean` within the configuration for EditLive! for XML.

Populating the Styles Drop-down List Box

When style information is added to EditLive! for XML then style information will be added to the style drop-down list box. This means that users can quickly and easily access style information. For example, if a style sheet defined the following styles:

```
h1{color: yellow}
p.blue{color: blue}
span.red{color: red}
.green{color: green}
```

Then *.blue* and *.red* and *.green* would appear as options within the styles drop-down list box giving users easy access to this style information. Also the *Heading 1* style would have yellow colored text.

In the EditLive! for XML styles drop down block styles are depicted with a ¶ symbol next to them and inline styles are depicted with a **a** symbol next to them. Thus, the style drop down for the above example may appear as follows:

Heading 1 ¶

.blue ¶

.red a

green ¶

.green a

Note that the `.green` class appears twice as it can be applied as both an inline and a block style. For more information please see the Specifying Inline and Block Styles section of this article.

Summary

EditLive! for XML provides excellent support for implementing cascading style sheets (CSS). Style information can be used with an EditLive! for XML document in a number of ways; as an externally linked style sheet, as an embedded style sheet or importing content from Microsoft Word. This style information will then be used by EditLive! for XML to populate the styles drop-down list box giving users easy access to style information.

See Also

- `<link>` element
- `<style>` element
- `<wordImport>` element
- W3C's Introduction to CSS [http://www.w3.org/MarkUp/Guide/Style]

Using Ephox CSS Extensions with Custom Tags

EditLive! for XML includes extensive support for the embedding of custom tags within HTML. As part of the support of EditLive! for XML for custom tags in HTML developers can provide custom rendering for these tags. Ephox CSS extensions can are used to customize the rendering of these tags. The styles with which custom tag rendering is specified in EditLive! for XML can be provided either as an external, linked style sheet or through an embedded style sheet. For more information on using style sheets with EditLive! for XML please see the section on Ephox EditLive! for XML and CSS.

Note

Inline style information should not be used to specify rendering for a custom tag.

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.

Example 6.1. Declaring a CSS Style for Rendering a Custom Tag

The following example demonstrates how to specify rendering information for the custom tag `<MyTag>` and its closing tag `</MyTag>`. The icon `myicon.jpg` (mapped to by the URL `http://www.server.com/icons/myicon.jpg`), 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.

When specifying a custom tag to be used with EditLive! for XML the `display` attribute must be assigned. The value of the `display` attribute affects the way in which the custom tag is processed by EditLive! for XML. For more information see the reference on the `display` attribute.

```
MyTag{
  display: block;
  ephox-icon: url(http://www.server.com/icons/myicon.jpg);
  ephox-start-label: Custom Tag;
  ephox-end-label: /Custom Tag
}
```

More Information

For more information on the Ephox CSS Extensions for Custom Tags please see the chapter titled Ephox CSS Extensions for Custom Tags.

Chapter 7. Deployment Optimizations

Minimizing an EditLive! for XML Deployment

It is possible to minimize a deployment of EditLive! for XML which can be especially useful when deploying EditLive! for XML to Web servers with a limited amount of disk space available.

All the files required to deploy EditLive! for XML to a Web server are included in the *`INSTALL_HOME`*`/webfolder/redistributables` directory of your EditLive! for XML install where *`INSTALL_HOME`* is the directory to which the EditLive! for XML SDK has been installed.

The `redistributables` directory contains three sets of files:

- The EditLive! for XML source files, these are found in the `.../redistributables` directory.
- The EditLive! for XML dictionaries, there are found in the `.../redistributables/dictionaries` directory

In the `redistributables/dictionaries` it is possible to remove all the dictionaries apart from those being used with your implementation of EditLive! for XML. The dictionary to be used with an instance of EditLive! for XML is specified within the EditLive! for XML configuration XML in the **<spellCheck>** element.

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 **LocalDeployment** property of your SDK must be set to `false` when instantiating EditLive! for XML.

Summary

In order to minimize the install of EditLive! for XML the developer can remove various files from the `redistributables` directory and its subdirectories. When removing files from this directory the developer should take care to ensure that the files are not in use in their deployment of EditLive! for XML.

See Also

- **<spellCheck>** XML element

Optimizing EditLive! for XML Load Times

Introduction

This document discusses ways in which the load time of EditLive! for XML can be optimized. There are several ways in which the load time of EditLive! for XML may be reduced. These include preloading the Java Plug-in, configuring the instance of EditLive! for XML 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! for XML 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! for XML 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! for XML 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! for XML within the Web application during the relevant session.

An example of the preloading functionality, implemented in JavaScript, can be seen below. For more information please refer to the Preload property in the EditLive! for XML SDK.

Example

The following code would preload the JVM and EditLive! for XML classes and then alert the user that EditLive! for XML has finished loading by using the `preloadReturn` callback function to create a JavaScript **alert** dialog.

```
<script language="javascript">
  ...
  var editlive1;
  editlive1 = new EditLiveXML("ELApplet1","1","1");
  editlive1.setDownloadDirectory("redistributables/editlivexml");
  editlive1.setConfigurationFile("editlivexml/sample_elconfig.xml");
  editlive1.setBody("&nbsp;"); editlive1.setPreload("preloadReturn");
  editlive1.show();
  function preloadReturn(){
    alert("Preloading of the JRE and applet is complete.");
  }
  ...
</script>
```

Configuring EditLive! for XML via ConfigurationText

When using an XML configuration file to customize the EditLive! for XML interface EditLive! for XML must make a HTTP request to the server to retrieve the relevant XML file. However, if the EditLive! for XML configuration is set via an XML document which is embedded directly in the relevant page EditLive! for XML no longer has to make an extra HTTP request to the Web server and load time is decreased.

Embedding an XML 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! for XML. This is achieved via the ConfigurationText property.

Example

The following JavaScript code passes in an XML document which will be used to customize EditLive! for XML. The code uses the default JavaScript **escape** function to encode the document, however, where possible, a server-side URL encoding method should be used to escape the XML text.

Note

- The XML document seen here is not complete. The XML text passed in must comply with the EditLive! for XML Configuration DTD.
- The string passed to the **setConfigurationText** property must be

URL encoded or encoded using the JavaScript **escape** function. The example below uses the JavaScript **escape** function but it is recommended that a server-side URL encoding method be used.

```
var editlive1;
editlive1 = new EditLiveXML("ELApplet1","700","400");
editlive1.setConfigurationText(
  escape('<?xml version="1.0" encoding="UTF-8"?> <editLive>...');
```

Setting the Data Model, Views and XML Document

The data model and views for EditLive! for XML and the Visual Designer can also be provided by text embedded in the Web page. Views can be added to EditLive! for XML through the **addViewAsText** method which data models may be added via the **addXSDAsText**. Loading views and data models via this method will decrease load times by ensuring that EditLive! for XML is not required to make a request to the server. Views and data models loaded via this method must be provided as a complete, URL encoded, XML style sheet (XSL) or XML Schema Documents (XSDs) respectively. It is advised that the file reading and URL encoding capabilities of a server-side scripting language be used to provide the XSL to the relevant Web page.

Example

The following JavaScript code passes in an XSL and an XSD which will be used to by EditLive! for XML. The code uses the default JavaScript **escape** function to encode the documents, however, where possible, a server-side URL encoding method should be used to escape the document text.

Note

- The XSL and XSD documents seen here are not complete. The document text passed in must comply with the XSL and XSD standards.
- The document text passed to the **addViewAsText** and **addXSDAsText** methods must be URL encoded or encoded using the JavaScript **escape** function. The example below uses the JavaScript **escape** function but it is recommended that a server-side URL encoding method be used.

```
var editlive1;
editlive1 = new EditLiveXML("ELApplet1","700","400");
editlive1.addViewAsText(
  escape('<?xml version="1.0"?>...','First View'));
editlive1.addXSDAsText(
  escape('<?xml version="1.0"?>...'));
```

Deploying the Java Runtime Environment

In order to use EditLive! for XML 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 for Java 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 XML for this purpose. To use this installer simply set the local deployment property of EditLive! for XML to `true`. The following example demonstrates how to achieve this with the LocalDeployment property of the JavaScript SDK.

Example

The following code sets EditLive! for XML to use the local copy of the JRE files from the server.

```
editlive1.setLocalDeployment(true);
```

Summary

The load time of EditLive! for XML can be optimized through the use of several developer features. Through the use of these features initial load times can be reduced. Developers can optimize the load time of EditLive! for XML by preloading the Java Plug-in, setting various properties via text embedded in the relevant Web page and downloading the JRE from the local servers if EditLive! for XML is being accessed via an intranet.

See Also

- Preload property

- ConfigurationText property
- LocalDeployment property

Chapter 8. Customizing EditLive! for XML

Creating Custom Dictionaries for Ephox EditLive! for XML

Introduction

Custom dictionaries may be created for use with Ephox EditLive! for XML. Custom dictionaries are used to add words to an existing dictionary used by the spell checker for EditLive! for XML. This document details how to extend the existing dictionary of a spell checker used by EditLive! for XML. This requires some skill with unzipping files and using the Java **jar** command.

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.

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 `webfolder/redistributables/editlivexml` directory of your EditLive! for XML install. The name of this .jar file will vary with the spell checker your version of EditLive! for XML is using. Typically the name of the .jar file will be capitalized. If in doubt as to the name of your existing dictionary please see either the document on Using Different Spell Checkers with Ephox EditLive! for XML or the **<spellCheck>** XML element.

   For the purposes of this document and the example contained herein it will be assumed that the spell checker concerned is named `en_us_3_0.jar`.

   Once you have found this file use a file unzipping utility to extract the contents of the file. In order to complete the steps required to create a custom dictionary the spell checker file should be unzipped to a new directory which, initially, contains only the content extracted from the .jar file.

2. If the file has been extracted correctly from the zip file there should now be two directories in the directory to which the file was unzipped. Note that these directories should currently be the only listings within this directory. These

directories are `com` and `META-INF`. For the creation process to proceed correctly delete the `META-INF` directory from the directory.

Note

If a custom dictionary has already been created for this spell checker there will also be a file named `userdic.tlx` present in the directory to which the .jar file was extracted. If this is the case and you wish to extend the existing custom dictionary then **do not** delete this file. For more information on this please see the Modifying the Current Custom Dictionary section of this document. If you wish to remove the current custom dictionary then delete this file. For more information on this 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 directory to which 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 new 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". 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 to which the contents of the original .jar file were extracted.

   For example, if the location of the directory to which the contents of the original .jar file were extracted was *`C:\OriginalJar\`*, the location of the **jar** command was `C:\java\bin\jar` and you wished to name the new .jar file `customdict.jar` then the command line would be as follows:

   ```
   C:\java\bin\jar cvf customdict.jar .
   ```

Note

This command is as it would appear if you were currently browsing the directory to which the original .jar file was extracted via the command prompt.

8. Move the newly compiled .jar file, in the case of the example above the file would be called `customdict.jar` to a location where it may be accessed by the EditLive! for XML applet.
9. Edit the configuration information for EditLive! for XML to reflect the location of the new spell checker. For more information on how to do this see the Setting the Spell Checker for EditLive! for XML section of this document, the document on Using Different Spell Checkers with Ephox EditLive! for XML or the **<spellCheck>**XML element.

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 to which the contents of the original .jar file were extracted. The name of the new spell checker .jar file is specified in this step.

   For example, if the location of the directory to which the contents of the original .jar file were extracted was `C:\OriginalJar\`, the location of the **jar** command was `C:\java\bin\jar` and you wished to name the new .jar file `customdict.jar` then the command line would be as follows:

```
C:\java\bin\jar cvf customdict.jar .
```

Note

This command is as it would appear if you were currently browsing the directory to which the original .jar file was extracted via the command prompt.

6. Move the newly compiled .jar file, in the case of the example above the file would be called `customdict.jar` to a location where it may be accessed by the EditLive! for XML applet or save it in place of the old file.

   If the name of the new custom dictionary differs from the old one then edit the configuration information for EditLive! for XML to reflect the location of the new spell checker. For more information on how to do this see the Setting the Spell Checker for EditLive! for XML section of this document, the document on Using Different Spell Checkers with Ephox EditLive! for XML or the **<spellCheck>**XML element.

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 to which 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 to which the contents of the original .jar file were extracted. The name of the new spell checker .jar file is specified in this step.

   For example, if the location of the directory to which the contents of the original .jar file were extracted was *`C:\OriginalJar\`*, the location of the **jar** command was `C:\java\bin\jar` and you wished to name the new .jar file `customdict.jar` then the command line would be as follows:

```
C:\java\bin\jar cvf customdict.jar .
```

Note

This command is as it would appear if you were currently browsing the directory to which the original .jar file was extracted via the command prompt.

4. Move the newly compiled .jar file, in the case of the example above the file would be called `customdict.jar` to a location where it may be accessed by the EditLive! for XML applet or save it in place of the old file.

   If the name of the new custom dictionary differs from the old one then edit the configuration information for EditLive! for XML to reflect the location of the new spell checker. For more information on how to do this see the Setting the Spell Checker for EditLive! for XML section of this document, the document on Using Different Spell Checkers with Ephox EditLive! for XML or the **<spellCheck>**XML element.

Setting the Spell Checker for EditLive! for XML

The spell checkers for EditLive! for XML for different languages are each packaged as separate files. In order to configure EditLive! for XML to use a spell checker the program must be informed of the location of the relevant spell checking package, this occurs via the EditLive! for XML configuration. The **<spellCheck>** element of the EditLive! for XML configuration provides EditLive! for XML with the location of the spell checker package which it is to use. The location is specified as a URL which can be either relative or absolute. For more information please see the article on Using Different Spell Checkers with Ephox EditLive! for XML.

Example

This is an example of a full `userdic.tlx` file that can be used to extend a current dictionary.

```
#LID 24941
customdict    i
Ephox    i
EditLive    i
Java    i
HTML    i
XML    i
```

Summary

The dictionary to be used with EditLive! for XML can be extended to include words which are not in the standard dictionary. This is done by including a file called `userdic.tlx` in the .jar file of the spell checker which is used by EditLive! for XML. The included file should contain all the words which are to be added to the EditLive! for XML spell checker.

See Also

- **<spellCheck>** XML element
- Using Different Spell Checkers with Ephox EditLive! for XML

Customizing the EditLive! for XML Interface

Introduction

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

The Menu Bar

The menu bar in EditLive! for XML 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.

For example, this would specify the `View` menu which has the mnemonic of "V":

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

Menu Items

Menu items within EditLive! for XML are specified through the use of a **<menuItem>** XML element. The **name** attribute of the **<menuItem>** element determines which menu item is inserted. EditLive! for XML is provided with a collection of predefined interface items which may be placed on the EditLive! for XML menus, the shortcut menu or the toolbars. Items from the interface command collection will have default menu item text associated with them. They may also have default mnemonics, images and shortcuts. For more information on what interface commands are available for use please see the EditLive! for XML Interface Command Collection section of this document.

The following example would add the **New**, **Open...**, **Save** and **Save As...** items to a menu with the name **File**.

```
<editLive>
  ...
  <menuBar>
    ...
      <menu name="&amp;File">
        <menuItem name="New"/>
        <menuItem name="Open"/>
        <menuItem name="Save"/>
        <menuItem name="SaveAs"/>
      </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! for XML are specified through the use of a **<menuItemGroup>** XML element. The **name** attribute of the **<menuItemGroup>** element determines which menu item group is inserted. EditLive! for XML is provided with a collection of predefined interface item groups which may be placed on the EditLive! for XML menus and toolbars. Items from the interface command collection will have default menu item text associated with them.

They may also have default mnemonics, images and shortcuts. For more information on what interface commands are available for use please see the EditLive! for XML Interface Command Collection section of this document. 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.

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

```
<editLive>
  ...
  <menuBar>
    ...
      <menu name="&amp;View">
        <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>** within a **<menu>** element. They serve no purpose other than that of a visual aid.

The About Dialog

The Ephox logo:

and associated **About** dialog may be removed from the menu interface. The removal of this menu option is achieved through setting the **showAboutMenu** attribute of the **<menuBar>** element to `false`.

For example, the following XML would remove the Ephox branding and associated **About** dialog from the menu:

```
<editLive>
  ...
  <menuBar showAboutMenu="false">
    ...
  </menuBar>
  ...
```

```
</editLive>
```

Toolbars

The EditLive! for XML applet can be instantiated with multiple toolbars. The **`<toolbar>`** element is used to specify a toolbar within the EditLive! for XML configuration file. Toolbars will appear within EditLive! for XML in the order 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 user interface of EditLive! for XML.

The following XML 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! for XML are specified through the use of a <toolbarButton> XML element. The **name** attribute of the <toolbarButton> element determines which toolbar button is inserted. EditLive! for XML is provided with a collection of predefined interface items which may be placed on the EditLive! for XML toolbars, the menus or shortcut menu. 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 commands without associated images are not placed on the toolbars. For more information on what interface commands are available for use please see the EditLive! for XML Interface Command Collection section of this document.

The following 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! for XML are specified through the use of a **`<toolbarButtonGroup>`** XML element. The **name** attribute of the **`<toolbarButtonGroup>`** element determines which toolbar button group is inserted. EditLive! for XML is provided with a collection of predefined interface item groups which may be placed on the EditLive! for XML 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 XML Interface Command Collection section of this document.

The following 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>
```

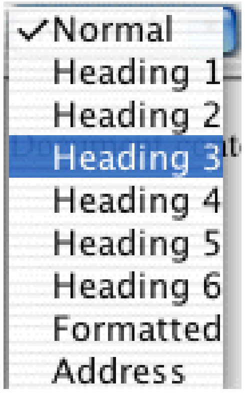
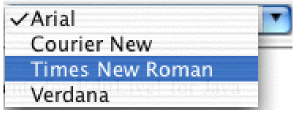
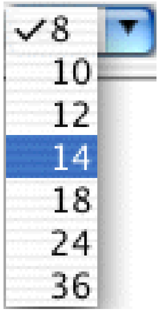
Toolbar Combo Boxes

Combo boxes for the style, font typeface and font size text attributes can be added to the EditLive! for XML toolbar through the use of a **<toolbarComboBox>** 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.

For example, the following XML would create the **Style** drop down combo box in the EditLive! for XML **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="tlbStyle">
          <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! for XML, their corresponding function and their associated value for the `<toolbarComboBox>` **name** are listed below.

Example Image	**Function**	**XML Name Attribute**
 ✓Normal Heading 1 Heading 2 Heading 3 Heading 4 Heading 5 Heading 6 Formatted Address	List of styles available for use in this document.	tlbStyle
	List of fonts available for use in this document.	tlbFace
	List of font sizes available for use in this document.	tlbSize

Toolbar Separators

Toolbar separators are vertical lines spanning the height of the toolbar which can be used to visually break a toolbar into its constituent parts and areas. These are added through the use of the **<toolbarSeparator>** within a `<toolbar>` element. They serve no purpose other than that of a visual aid.

The Shortcut Menu

Shortcut menu items in EditLive! for XML are specified through the use of a `<shrtMenuItem>` XML element with a specific value for the **name** attribute associated with it. EditLive! for XML is provided with a collection of predefined interface items which may be placed on the EditLive! for XML shortcut menu. Items from the interface command collection will have default menu item text associated with them.

They may also have default images. The shortcut menu can also contain submenus. For more information on what interface commands are available for use please see the EditLive! for XML Interface Command Collection section of this document.

Note

Menu item groups cannot be used on the shortcut menu.

The following example would add the **Cut**, **Copy** and **Paste** items to the **Shortcut menu**:

```
<editLive>
  ...
  <shortcutMenu>
      <shrtMenu>
        <shrtMenuItem name="Cut"/>
        <shrtMenuItem name="Copy"/>
        <shrtMenuItem name="Paste"/>
      </shrtMenu>
  </shortcutMenu>
  ...
</editLive>
```

The Element Menu

Within EditLive! for XML the Element Menu is the right-click menu associated with the **Document Navigator Bar**. Element menu items are specified through the use of a **`<elementMenuItem>`** XML element with a specific value for the **name** attribute associated with it. EditLive! for XML is provided with a collection of predefined interface items which may be placed on the element menu. Items from the interface command collection will have default menu item text associated with them. They may also have default images. For more information on what interface commands are available for use please see the EditLive! for XML Interface Command Collection section of this document.

Note

Menu item groups cannot be used on the element menu.

The following example would add the **Insert Before**, **Insert After** and **Insert Into** items to the **Element menu**:

```
<editLive>
```

```
  ...
  <shortcutMenu>
      <elementMenu>
        <elementMenuItem name="xmlInsertBefore"/>
        <elementMenuItem name="xmlInsertAfter"/>
        <elementMenuItem name="xmlInsertAtCurrent"/>
      </elementMenu>
  </shortcutMenu>
  ...
</editLive>
```

Submenus

Submenus can be added to the EditLive! for XML menu bar and shortcut menu. The submenus available for use in EditLive! for XML are the **Font**, (font) **Size** and **Style** submenus. These are added through the use of the **<submenu>** XML element with a specific value for the **name** attribute.

The **<submenu>**, if left empty, each of the submenus added will contain the same items as the corresponding item on the EditLive! for XML 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. It should be noted that once menu items are specified within a submenu then the contents of the submenu will no longer mirror the corresponding toolbar element.

The following example would add the **Font** submenu to the **Format** menu with items on the submenu corresponding to the items specified in the **Font** drop down combo box of the EditLive! for XML toolbar:

```
<editLive>
  ...
  <menuBar>
    ...
    <menu name="Format">
      ...
      <submenu name="mnuFontFace"/>
      ...
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

The following 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! for XML toolbar:

```
<editLive>
  ...
  <shortcutMenu>
    <shrtMenu>
      ...
      <submenu name="mnuFontFace" />
      ...
    </shrtMenu>
  </shortcutMenu>
  ...
</editLive>
```

The following example would add the **Style** submenu to the **Format** menu. The menu created would contain the **Normal** and **Heading 1** 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="Format">
      ...
      <submenu name="mnuFontStyle">
        <menuItem name="P" text="Normal" />
        <menuItem name="H1" text="Heading 1" />
      </submenu>
      ...
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

The submenus available for use with EditLive! for XML, their corresponding toolbar item, mnemonic and their associated value for the `<submenu>` **name** are listed below.

Submenu Item Name	**XML Name Attribute**	**Corresponding Toolbar Item**	**Mnemonic**

Font	mnuFontFace	tlbFace - Font	F
Size	mnuFontSize	tlbSize - Size	S
Style	mnuStyle	tlbStyle - Style	T
Color	mnuColor	N/A	C
Highlight Color	mnuHighlightColor	N/A	C

Note

The **Color** and **Highlight Color** submenus should only be used when customizing the **Color** and **Highlight Color** menu items. For more information on customizing the color choosers in EditLive! for XML see the section on Customizing the Color Choosers.

EditLive! for XML Interface Command Collection

Menu and Toolbar Button Items

This collection of interface commands can be used within menus and toolbars in EditLive! for XML. The items, their corresponding function, tool tip and menu text, mnemonic, shortcuts, images and their associated value for the relevant **name** attribute are listed below:

Note

It is recommended that items, such as **Save As...**, are not used within toolbars as they do not have a default image associated with them.

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
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
Select all editor content.	SelectAll	Select All	CTRL+A	N/A	L
Find text in the editor.	Find	Find...	CTRL+F	🔍	F
Insert a hyperlink.	HLink	Hyperlink...	CTRL+K	🔗	H
Insert a horizontal line.	HRule	Horizontal Line	N/A	☰	L
Insert a symbol.	Symbol	Symbol...	N/A	Ω	S
Insert a bookmark.	Bookmark	Bookmark...	N/A	🔖	K
Insert an image	ImageLocal	Local	N/A	N/A	M

from the local machine.		Image...			
Insert an image from the server image library.	ImageServer	Server Image Library...	N/A		I
Insert a HTML comment.	InsertComment	Insert Comment...	N/A	N/A	N/A
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 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

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
Toggle table gridlines on and off.	Gridlines	Show/Hide Gridlines	N/A	▦	G
Run the spell checker.	Spelling	Spelling...	F7	ABC✓	S
Perform a word count on the document.	WordCount	Word Count...	N/A	ℹ	W
The text color selector.	Color	Color	N/A	T	C
The text highlight color selector.	HighlightColor	Highlight Color	N/A	ab	C
Bold text.	Bold	Bold	CTRL+B	**B**	B
Italic text.	Italic	Italic	CTRL+I	*I*	I
Underline text.	Underline	Underline	CTRL+U	U	U
Strikethrough text.	Strike	Strikethrough	N/A	~~AB~~	S

Remove formatting.	RemoveFormatting	Remove Formatting	N/A		R
Increase the paragraph or list indent.	IncreaseIndent	Increase Indent	N/A		N
Decrease the paragraph or list indent.	DecreaseIndent	Decrease Indent	N/A		D
Edit the properties of a list.	PropList	List Properties...	N/A	N/A	N/A

EditLive! for XML Specific Menu and Toolbar Button Items

The following menu and toolbar items can only be added to the interface within EditLive! for XML.

Function	**XML Name Attribute**	**Menu or Tool Tip Text**	**Shortcut**	**Image**	**Mnemonic**
Insert a new XML element before the current element. This is useful within sections with repeating elements.	xmlInsertBefore	Insert Before	N/A		N/A
Insert a new XML element after the current element. This is useful within sections with repeating elements.	xmlInsertAfter	Insert After	N/A		N/A

Insert a new XML element at the location of the current element. This is useful within sections with repeating elements.	xmlInsertAtCurrent	Insert Into	N/A	↪	N/A
Convert the current XML element into another, valid, XML element.	xmlConvert	Convert Element	N/A	⇄	N/A
Add an attribute to the selected XML element. Only valid elements can be added.	xmlAddAttribute	Add Attribute	N/A	N/A	N/A
Move the current XML element up one position within the document. This is useful within sections with repeating elements.	xmlMoveUp	Move Up	N/A	⬆	N/A
Move the current XML element down one position within the document. This is useful within sections with repeating elements.	xmlMoveDown	Move Down	N/A	⬇	N/A

Remove the current XML element. This is useful within sections with repeating elements.	xmlRemove	Remove	N/A	❌	N/A

Visual Designer Specific Menu and Shortcut Menu Items

The following items can only be placed on the menus and shortcut menu of the Visual Designer. These menu items must be placed within the `<designerMenuItem>` element of the configuration file.

Function	**XML Name Attribute**	**Menu or Tool Tip Text**	**Shortcut**	**Image**	**Mnemonic**
Edit the properties of the currently selected form control.	ControlProperties	Control Properties...	N/A	N/A	N/A
Export the current view as an XML StyleSheet (XSL).	ExportXSLT	Export StyleSheet...	N/A	N/A	N/A
Export the data model as an XML Schema Document (XSD).	ExportXSD	Select All	N/A	N/A	N/A

Menu Item and Toolbar Button Groups

This collection of interface commands can be used only within menus and toolbars in EditLive! for XML. These interface items are added as a group of buttons or menu items within EditLive! for XML. 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:

Note

These groups cannot be added to the shortcut or element menus within EditLive! for XML.

Function	XML Name Attribute	Menu or Tool Tip Text	Shortcut	Image	Mnemonic
View the document in Design mode (WYSIWYG mode).	SourceView	Design View	N/A	N/A	D
View and edit the HTML source for the document.		HTML View	N/A	N/A	H
View the applet in the browser.	FrameView	Browser View	N/A	N/A	B
View the applet in a separate window.		Window View	N/A	N/A	W

Insert an ordered list or change an unordered list to an ordered list.	List	Ordered List	N/A	☰	O
Insert an unordered list or change an ordered list to an unordered list.		Unordered List	N/A	☰	T
Set left alignment.	Align	Align Left	CTRL+L	☰	L
Set center alignment.		Align Center	CTRL+E	☰	C
Set right alignment.		Align Right	CTRL+R	☰	R
Superscript text.	Script	Superscript	N/A	x^2	S
Subscript text.		Subscript	N/A	x_2	S

Customizing Available Items

The interface items available through the standard EditLive! for XML 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>`** and the **text** and **imageURL** attributes of the **`<toolbarButton>`** element.

Note

The properties of menu item and toolbar button groups cannot be customized.

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="&amp;File">
        <menuItem
          name="New"
          text="Create New"
          imageURL="customImage.gif"
          mnemonic="c"
        />
        ...
      </menu>
    ...
  </menuBar>
  ...
</editLive>
```

Customizing the Color Choosers

The color choosers for both the **Color** and **Hightlight Color** menu and toolbar items can be customized to include only a specific set of predefined colors. The **More Colors...** item will also be available in addition to any predefined colors.

In order to customize the **Color** or **Highlight Color** menu items the relevant `<menuItem>` element must be replaced with a `<submenu>` element with the same `name` attribute. Each color should then be listed as a separate `<menuItem>` element. For each color `<menuItem>` element the `name` attribute should provide the HTML color value (either as a hexidecimal color value or reserved color key word, e.g. `#FF0000` or `red`). The `text` attribute for each color `<menuItem>` element should provide a text description of the color.

Example 8.1. Custom Color Chooser in a Submenu

The following example creates a custom color chooser in a submenu to customize the **Color** menu item. The submenu will contain the colors **Red**, **Blue** and **Green**.

```
<editLive>
  ...
  <menuBar>
    ...
      <menu name="F&amp;ormat">
```

```
        ...
        <submenu name="mnuColor">
          <menuItem name="#FF0000" text="Red" />
          <menuItem name="blue" text="Blue" />
          <menuItem name="green" text="Green" />
        </submenu>
        ...
      </menu>
    ...
  </menuBar>
  ...
</editLive>
```

In order to customize the **Color** or **Highlight Color** toolbar items the relevant `<toolbarButton>` element must be replaced with a `<toolbarComboBox>` element with the same `name` attribute. Each color should then be listed as a separate `<comboBoxItem>` element. For each color `<comboBoxItem>` element the `name` attribute should provide the HTML color value (either as a hexidecimal color value or reserved color key word, e.g. `#FF0000` or `red`). The `text` attribute for each color `<comboBoxItem>` element should provide a text description of the color.

Example 8.2. Custom Color Chooser in a Toolbar

The following example creates a custom color chooser in a submenu to customize the **Highlight Color** toolbar button. The drop down will contain the colors **Red**, **Blue** and **Green**.

```
<editLive>
  ...
  <toolbars>
    ...
      <toolbar name="format">
        ...
        <toolbarComboBox name="tlbHighlightColor">
          <comboBoxItem name="#FF0000" text="Red" />
          <comboBoxItem name="blue" text="Blue" />
          <comboBoxItem name="green" text="Green" />
        </toolbarComboBox>
        ...
      </toolbar>
    ...
  </toolbars>
  ...
</editLive>
```

Creating New Items

Custom items can be added to the EditLive! for XML menus and toolbars. This gives the developer greater flexibility when integrating EditLive! for XML in existing systems. It allows developers to complement the functionality of EditLive! for XML with existing JavaScript functions and also to extend the menus and toolbars of the EditLive! for XML applet to include custom functionality.

EditLive! for XML allows for the specification of custom menu items, custom toolbar buttons and custom toolbar combo boxes, this can be done with the **<customMenuItem>**, **<customToolbarButton>** and **<customToolbarComboBox>** elements respectively.

When creating custom items to use with EditLive! for XML setting the **xhtmlonly** attibute to `true` ensures that the custom interface item will only be enabled whilst the cursor is within an XHTML section.

For more information on how to use custom items in EditLive! for XML please see the article on Custom Menu and Toolbar Items for EditLive! for XML.

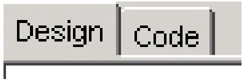
Tab Views

EditLive! for XML 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! for XML editing area, they can also be removed completely. The configuration of the tabbed view for EditLive! for XML is achieved via the **tabPlacement** attribute of the **<wysiwygEditor>** element.

For example, the following XML would place the view tabs at the top of the EditLive! for XML editing pane.

```
<editLive>
    ...
    <wysiwygEditor tabPlacement="top">
    ...
</editLive>
```

The tabbed view settings for use with EditLive! for XML and their associated value for the `<wysiwygEditor>` **tabPlacement** attribute are listed below.

Example Image	**Function**	**XML tabPlacement Attribute**
	Place tabs at the top of the editing pane.	top
	Place tabs at the bottom of the editing pane.	bottom
	Remove tabs.	off

Removing the Menu Bar and Toolbars

Both the menu bar and either of the toolbars of EditLive! for XML may be removed. In order to display EditLive! for XML without any toolbars ensure that there are no **`<toolbar>`** elements within the configuration file.

To remove the menu bar ensure that the **<menuBar>**element is empty (i.e. it has no child elements) and ensure that the **showAboutMenu** attribute of the **<menuBar>**element is set to `false`.

Limiting the Functionality of EditLive! for XML

Removing specific functionality from EditLive! for XML is achieved by removing the corresponding menu and/or toolbar buttons from the EditLive! for XML interface. When the item is not included in the XML configuration then the item will not appear on the menu or toolbar and, in most cases, the shortcut key for the item will be disabled.

It should be noted that 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.

Summary

The interface for EditLive! for XML is highly customizable. Through the EditLive! for XML configuration process developers gain the flexibility to make the interface for EditLive! for XML as simple or as complex as they wish. EditLive! for XML is supplied with a standard complement of menu and toolbar items providing editor functionality. These items are listed in the above sections of this document along with all the relevant information pertaining to each item.

Furthermore, EditLive! for XML gives the developer the ability to create custom functionality which can be accessed through custom menu and toolbar items. This can be used to complement the functionality of the EditLive! for XML applet with the developers own custom functions and macros.

Finally, through the exclusion of relevant menu and toolbar items the developer can limit the functionality of the EditLive! for XML applet, thus preventing end users from accessing functionality which the developer does not wish them to.

Custom Menu and Toolbar Items for EditLive! for XML

Introduction

Ephox EditLive! for XML allows for the developer to specify custom items on the toolbars and menus. These custom items can be used to insert specific HTML source or a hyperlink at the cursor location, it can also be used to raise a JavaScript event or cause EditLive! for XML to POST its content to a specific URL. For information on how to configure EditLive! for XML to include custom menu and toolbar items see the **<customToolbarButton>**, **<customToolbarComboBox>** and **<customMenuItem>** elements in the EditLive! for XML Configuration Reference.

Within EditLive! for XML custom menu items and toolbar buttons which insert hyperlinks or HTML at the cursor will function only within XHTML sections. Those custom functions which use the raise event functionality will function throughout the editor.

Custom Toolbar Options

There are two types of custom items which can be added to the EditLive! for XML toolbar, these are custom toolbar buttons and custom toolbar combo items. Custom toolbar buttons appear as buttons within the toolbar and when configuring these within the EditLive! for XML configuration file a URL which corresponds to an image for the button must be included. This image is used on the button when it is placed in the toolbar. Custom combo boxes appear in the same format as the typeface, style and font size toolbar items. When configuring custom combo boxes for use within EditLive! for XML each individual item within the combo box must be configured via the **<customComboBoxItem>**element in the EditLive! for XML configuration file.

Examples

The following example demonstrates how to define a **custom toolbar button** for use within EditLive! for XML on the **Command Toolbar**. 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.

Example 8.3. Configuring a Custom Toolbar Button

```
<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>
```

The following example demonstrates how to define a custom combo box item for use within a **custom toolbar combo box** which exists on the EditLive! for XML **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.

Example 8.4. Configuring a Custom Toolbar Combo Box

```
<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>
```

Custom Menu Items

Custom menu items in EditLive! for XML can only be of one format. Custom menu items can only be of a single layered depth (i.e. they cannot include submenus). Custom menu items can be added to any of the menus in EditLive! for XML via the **<customMenuItem>**in the EditLive! for XML configuration file.

Example

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

Example 8.5. Configuring a Custom Menu Item

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <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>
```

Inserting HTML and Hyperlinks at the Cursor

Custom toolbar buttons and combo boxes, and custom menu items can be configured within EditLive! for XML to insert specific HTML source or hyperlinks at the location of the cursor. When configuring custom toolbar buttons, combo boxes and menu items within EditLive! for XML to insert HTML at the cursor the HTML source to be inserted at the cursor needs to be specified within the EditLive! for XML configuration file. When declaring the HTML to insert in the XML configuration file the HTML needs to be URL encoded [http://www.ietf.org/rfc/rfc2396.txt?number=2396].

The insert hyperlink at cursor custom functionality in EditLive! for XML 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 configuration of the custom item.

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 XML configuration file.

Examples

The following example demonstrates how to define a custom menu item which uses the **insertHTMLAtCursor** action for use within EditLive! for XML. The menu item defined in this example will insert `HTML to insert` at the cursor, note that the value in the example below is URL encoded.

Example 8.6. Configuring a Custom Item to Insert HTML at the Cursor

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <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>
```

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <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 menu item which uses the **insertHyperlinkAtCursor** action for use within EditLive! for XML. The menu item defined in this example will insert the URL `http://www.ephox.com` at the cursor.

Example 8.7. Configuring a Custom Item to Insert a Hyperlink at the Cursor

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <customMenuItem
        name="customItem2"
        text="Ephox"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="insertHyperlinkAtCursor"
        value="http://www.ephox.com"
      />
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

Enabling Custom Items Only in XHTML Sections

To ensure a custom interface item is only available whilst the cursor is placed within an XHTML section the **xhtmlonly** attribute of the relevant **<customToolbarButton>**, **<customToolbarComboBox>** or **<customMenuItem>** element must be set to `true`.

Raising a JavaScript Event

Custom toolbar and menu items in EditLive! for XML can be configured to raise JavaScript events. JavaScript events raised through EditLive! for XML are required to be defined either in the page in which EditLive! for XML is embedded or must be defined in a file which is included in the page in which EditLive! for XML is embedded.

When raising a JavaScript event from EditLive! for XML the **value** attribute of the related **<customComboBoxItem>**, **<customToolbarButton>** or **<customMenuItem>** element in the XML configuration file should specify 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! for XML. The menu item defined in this example will call the JavaScript function called `eventRaised`.

Example 8.8. Configuring a Custom Item to Use the raiseEvent Functionality

```
<editLive>
  ...
  <menuBar>
    ...
    <menu>
      <customMenuItem
        name="customItem1"
        text="Raise Event"
        imageURL="http://www.someserver.com/image16x16.gif"
        action="raiseEvent"
        value="eventRaised"
      />
    </menu>
    ...
  </menuBar>
  ...
</editLive>
```

Using Custom Properties Dialogs

The custom properties dialog functionality of EditLive! for XML allows developers to retrieve and set the attributes for a particular element within EditLive! for XML. Developers can use this functionality with their own JavaScript functions to manipulate the attributes of specific tags.

The custom properties dialogs should be linked to from a custom toolbar button or custom menu item. For more information on how to configure EditLive! for XML to include custom menu and toolbar items see the **<customToolbarButton>** and **<customMenuItem>** elements in the EditLive! for XML Configuration Reference.

Example 8.9. Configuring a Custom Item to Retrieve the Properties of a Tag

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

The JavaScript function corresponding to this custom menu item should be of the following form:

```
function customTDFunction(properties){
  ...
  //Content of function goes here
  ...
}
```

Where the parameter `properties` is a string.

POSTing the Content of EditLive! for XML

Custom toolbar and menu items in EditLive! for XML can be configured to cause EditLive! for XML to POST its content to a specific URL. When using the **PostDocument** function the **value** attribute of the custom item is used to pass several parameters to EditLive! for XML. These parameters are delimited by the string `##ephox##`. Thus, the string `editlive_field##ephox##http://someserver/post/POSTacceptor.aspx` contains two parameters `editlive_field` and `http://someserver/post/POSTacceptor.aspx`.

The **value** attribute for the **PostDocument** function may contain the following parameters.

POST Field	The name of the field in the HTTP POST that EditLive! for XML uses to POST its content. This parameter is required.
POST Acceptor URL	The URL for the POST acceptor that EditLive! for XML is to POST to. The parameter is required.
Response Processing	The operation that EditLive! for XML 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 repsonse processing is set to `callback`.

Note

The parameters must appear in the **value** attribute in the the order POST field, POST Acceptor URL, Response Processing, JavaScript Callback Function. Thus, the content of the **value** attribute may appear as follows:

For saving to disk:

`POST_field##ephox##http://someserver/postacceptor.jsp##ephox## saveToDisk`

`POST_field##ephox##http://someserver/postacceptor.jsp##ephox## callback##ephox##JSFunctionName`

Where *`POST_field`* is the name of the field the content is to be POSTed to, *`http://someserver/postacceptor.jsp`* is the URL for the POST acceptor script and *`JSFunctionName`* is the name of the JavaScript function to be used as a call back.

Example

The following example demonstrates how to define a custom menu item which uses the **PostDocument** action for use within EditLive! for XML. 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`.

Example 8.10. Configuring a Custom Item to Use the PostDocument Functionality

```
<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>
```

Summary

The developer can create extended customized functionality in EditLive! for XML via the custom toolbar and menu items. These items can be configured to insert a specific hyperlink or specific HTML at the cursor. They can also be configured to raise specific JavaScript events.

See Also

- Raising a JavaScript Event from Ephox EditLive! for XML
- Custom Properties Dialogs for EditLive! for XML
- Submitting EditLive! for XML Content Directly Via HTTP POST
- **<customToolbarButton>**
- **<customToolbarComboBox>**
- **<customMenuItem>**
- URL encoding (RFC 2396) [http://www.ietf.org/rfc/rfc2396.txt?number=2396]

Chapter 9. Using JavaScript for Customization

Custom Properties Dialogs for EditLive! for XML

Introduction

Ephox EditLive! for XML allows for developers to specify custom properties dialogs for use with specific tags. The custom properties dialog functionality of EditLive! for XML allows developers to retrieve and set the attributes for a particular element within EditLive! for XML. Developers can use this functionality with their own JavaScript functions to manipulate the attributes of specific tags.

The custom properties dialogs should be linked to from a custom toolbar button or custom menu item. For more information on how to configure EditLive! for XML to include custom menu and toolbar items see the **<customToolbarButton>** and **<customMenuItem>** elements in the EditLive! for XML Configuration Reference.

When to Use Custom Properties Dialogs

The custom properties dialogs functionality of EditLive! for XML can be used by developers when they wish to extend and customize the functionality of EditLive! for XML in respect to a specific tag. In this way developers may provide functionality which complements EditLive! for XML's base functionality.

Note

Custom properties dialogs in EditLive! for XML will function only within XHTML sections of the document.

Each menu or toolbar item related to a custom properties dialog is valid for use with a single tag as specified via the **enableintag** attribute of the relevant **<customToolbarButton>** or **<customMenuItem>** element. Thus, developers can create custom dialogs which pertain to specific tags. This is useful in providing new dialogs for HTML elements such as `<span>` which are not available for direct interaction within EditLive! for XML, to access the properties of custom or XML tags such as `<custom>`, or to replace or complement an existing EditLive! for XML dialog, such as the Image Properties dialog which corresponds with the `<img>` tag.

Interacting with EditLive! for XML and Custom Properties Dialogs

Interacting between EditLive! for XML and custom properties dialogs occurs via a JavaScript API. This API provides access to a list of relevant properties to a JavaScript function when a custom properties dialog is called from a toolbar or menu item within EditLive! for XML. The **SetProperties** function allows properties returned to EditLive! for XML.

JavaScript API for Custom Properties

Retrieving the Current Properties

In order to implement a custom properties dialog an associated toolbar or menu item must be created. This can be done within the XML configuration file for EditLive! for XML. The example below demonstrates how to create a custom menu item which can be used to access the properties for a `<td>` element. The example below calls the JavaScript function `customTDFunction`. This JavaScript function is supplied with a string that contains the name (or type) of the tag, a number which identifies the particular instance of the tag within the EditLive! for XML document and the existing attributes of the relevant tag. For more information on how to implement a custom properties dialog please see the Custom Properties Dialog function in the runtime API.

Example 9.1. Configuring a Custom Item to Retrieve the Properties of a Tag

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

The JavaScript function corresponding to this custom menu item should be of the following form:

```
function customTDFunction(properties){
  ...
  //Content of function goes here
  ...
}
```

Where the parameter `properties` is a string.

Setting New Properties

In order to return the changed properties to EditLive! for XML the SetProperties function of the JavaScript runtime API must be used. This function takes a single argument which is a string representing the name-value pairs for the relevant attributes. 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! for XML. 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).

Example 9.2. Using the SetProperties Function to Set Attributes

The following example sets the properties for a tag in the instance of EditLive! for XML named `ELApplet1` with the values stored in the `newProperties` string:

```
function makeChanges(paramOne,paramTwo){
  ...
  //Content of function goes here
  ELApplet1_js.SetProperties(newProperties);
  ...
}
```

When constructing the string variable to use with the **SetProperties** function care should be taken to ensure that the **ephoxTagID** and **tag** attributes are correct.

The ephoxTagID Attribute

The **ephoxTagID** attribute is used by EditLive! for XML to maintain a reference to the relevant tag inside EditLive! for XML. This attribute should not be changed when editing the properties of the tag. If the name or value of this attribute is altered in any way the custom properties dialog will not function correctly.

The tag Attribute

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! for XML. 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.

Standalone Attributes

The tag for which the custom properties dialog applies 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. EditLive! for XML outputs these attributes as a name-value pairing where the name and value are the same. In order to remove such attributes from the properties string both the relevant name and value strings should be removed from the properties string. 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.

Summary

EditLive! for XML includes functionality allowing developers to create their own custom properties dialogs for specific tags. This allows developers to complement and extend the existing functionality of EditLive! for XML with custom functions. When creating custom dialogs developers interact with EditLive! for XML via a JavaScript API. When interacting with the applet in this fashion it is important to remember several factors including the details of the **ephoxTagID** and **tag** attributes.

See Also

- Custom Menu and Toolbar Items for EditLive! for XML
- **<customToolbarButton>**
- **<customMenuItem>**
- URL encoding (RFC 2396) [http://www.ietf.org/rfc/rfc2396.txt?number=2396]

Raising a JavaScript Event from Ephox EditLive! for XML

Introduction

Ephox EditLive! for XML custom menu commands and custom toolbar buttons may be configured so that they raise a JavaScript event. This allows developers to complement the functionality of EditLive! for XML through the edition of their own JavaScript functions. Allowing for greater flexibility in the use of the EditLive! for XML applet.

Configuring EditLive! for XML to Raise JavaScript Events

The raising of JavaScript events from within EditLive! for XML can be associated with either a custom menu command or a custom toolbar button. In order to create a custom menu command or toolbar button with this functionality the EditLive! for XML configuration file must contain the relevant settings. For more information on the EditLive! configuration file see the Configuration Documentation.

Examples

The following example demonstrates how to configure an EditLive! for XML custom toolbar button to raise a JavaScript function called *javaScriptFunction*.

```
<editLive>
  ...
  <toolbars>
    <toolbar name="command">
      <customToolbarButton
        name="customButton1"
        text="Raise Event Button"
        imageURL="http://www.someserver.com/image20x20.gif"
        action="raiseEvent"
        value="javaScriptFunction"
      />
    </toolbar>
  </toolbars>
  ...
</editLive>
```

The following example demonstrates how to configure an EditLive! for XML custom menu command to raise a JavaScript function called `javaScriptFunction`.

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

```
        name="customItem1"
        text="Raise Event Command"
        imageURL="http://www.someserver.com/image20x20.gif"
        action="raiseEvent"
        value="javaScriptFunction"
      />
    </menu>
  </menuBar>
  ...
</editLive>
```

For more information on each of the XML attributes present in these tags please see the **<customToolbarButton>** or **<customMenuItem>** element documentation.

Using JavaScript Functions with Raise Event

In order to be able to use a JavaScript function with the EditLive! for XML raise event functionality the JavaScript function must included or defined in the same page as the instance of EditLive! for XML. The JavaScript function used should have no parameters.

Advanced Custom Functionality with Raise Event and EditLive! for XML

The raise event functionality of EditLive! for XML can be used in a variety of situations to complement the existing functionality of the EditLive! for XML applet. For example, if a custom toolbar or menu item within EditLive! for XML was to call a JavaScript function which, in turn, called the **window.open()** function the new window created, with the use of some program scripting, could be used to receive extra user input. Thus, a user may be able to select an image or file from a list presented in a new window, submit their selection back to another function in the parent page (the page in which the EditLive! for XML applet is embedded) and have the result of their selection placed into EditLive! for XML via the **InsertHTMLAtCursor** EditLive! for XML JavaScript API function.

Summary

Through the use of the raise event functionality of EditLive! for XML the functionality of the EditLive! for XML applet can be complemented by the developer through the use of JavaScript functions. This affords the developer a large degree of flexibility when using EditLive! for XML within their applications.

See Also

- **<customToolbarButton>**
- **<customMenuItem>**

Chapter 10. Image Upload

Using HTTP for Image Upload in Ephox EditLive! for XML

Overview

When inserting and uploading local images to a remote server, Ephox EditLive! for XML uses the HTTP multipart form-data protocol.

The `<httpImageUpload>` element of the XML configuration file contains the configuration information for the HTTP Image Upload functionality.

Why use HTTP?

Using the HTTP POST method to insert and upload images offers a secure way of allowing end users to interact with the remote server that the images are to be stored on.

Requirements for HTTP Image Upload

In order to upload local images to the remote server via HTTP, you will need a server-side image upload handler script that accepts the images on the server and stores them in the correct directory. 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 form element is used to upload the file to the server. This is the same script specified in EditLive! for XML to upload local images inserted into EditLive! for XML to the remote server.

EditLive! for XML HTTP Image Upload Configuration

The HTTP image upload configuration requires a URL corresponding to the image upload handler script and also a base property to use with image URLs. These properties can be set via the **href** and **base** attributes of the `<httpImageUpload>` element respectively. They can also be configured using the Configuration Tool. These settings are found on the **Image Settings** tab under the **HTTP Image Upload** settings.

HTTP Image Upload - href

This setting defines the location on the Web server of the script which handles image uploads.

Note

Relative URLs may be used. In this case the URL will be relative to the location of the page in which EditLive! for XML is embedded.

HTTP Image Upload - base

You should enter in this field the absolute URL for where the images are to be uploaded. The images can then be found by directing your browser to the images directory of the URL you supplied, after they have been uploaded.

Note

Relative URLs may be used. In this case the URL should be relative to the page which will be used to display the finished documents. For the images to be displayed correctly in the pages in which EditLive! for XML is embedded, the relative URL should also be the correct location relative to the pages in which EditLive! for XML is embedded.

Example 10.1. HTTP Image Upload Absolute BASE URL Configuration Example

In this example the location for the image upload script is `http://www.yourserver.com/scripts/upload.jsp`

The uploaded images can be found in a directory with the URL `http://www.yourserver.com/userimages/`

```
<editLive>
  ...
  <mediaSettings>
    <images>
      <httpImageUpload
        base="http://www.yourserver.com/userimages/"
        href="http://www.yourserver.com/scripts/uploadhandler.asp"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

Example 10.2. HTTP Image Upload Relative BASE URL Configuration Example

If the location of the display page directory is: `http://yourserver.com/editlivejava/startcms/site`

The location of the directory containing the pages with EditLive! for XML embedded in them is: `http://yourserver.com/editlivejava/startcms/cms`

Then the correct relative base URL setting for both the viewing and editing pages to function correctly with images would be: `../site/images`

The upload handler script is located at `http://yourserver.com/scripts/uploadhander.asp`

We recommend using absolute URLs when possible, as it is less likely to lead to confusion.

```
<editLive>
  ...
  <mediaSettings>
    <images>
      <httpImageUpload
        base="../site/images"
        href="http://yourserver.com/scripts/uploadhandler.asp"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

When are Images Actually Uploaded?

Images are uploaded when the contents of Ephox EditLive! for XML are submitted. This occurs when the form which contains Ephox EditLive! for XML has its **submit** method called.

What Form Field are Images Uploaded In?

Local images which are uploaded to the server by EditLive! for XML are placed within a form field with the name `image`. When implementing an upload acceptor script specifically to receive image files from EditLive! for XML the script should be made to accept files submitted within the `image` field.

Dynamically Setting the Image URL

It is possible to dynamically assign the URL for images embedded within the content as they are uploaded. This functionality is achieved via the POST acceptor for images. If the image POST acceptor returns a string containing a URL the URL is inserted into the document source code as the URL for the image. In this way it is possible to dynamically assign the directory images 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 string with the URL corresponding to the location for the uploaded image.

When setting the image URL in this manner the URL returned by the POST acceptor script to EditLive! for XML takes precedence over the **base** setting in the XML configuration file.

Note

The URL returned by the POST acceptor **must** be exactly the URL to be used for the image in EditLive! for Java.

Example Image Upload Scripts

EditLive! for XML is packaged with sample upload scripts for ASP, JSP and ASP.NET. 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! for XML SDK is installed.

Documentation is also available for the upload acceptor scripts:

- ASP Upload Script
- ASP.NET Upload Script
- JSP Upload Script

Common Problems

There are a number of problems that may occur while attempting to use HTTP Image Upload. These can be complicated and are chiefly to do with the settings on your server of choice.

- Sever side settings - ensure that the image 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 images are to be placed has write and read permissions set.

See Also

- Example ASP Upload Script
- Example ASP.NET Upload Script
- Example JSP Upload Script

ASP HTTP Image Upload Handler Script

Summary

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.

Defining the location of the image upload handler script

The location of the image upload handler script must be defined within the XML configuration file. This setting is configured via the **href** attribute of the `<httpImageUpload>` 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! for XML the **base** attribute of the `<httpImageUpload>` element must reflect the location of the directory where images on the Web server.

An example image upload handler script

Ephox 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! for XML SDK is installed.

Below are the steps required to use the ASP image upload handler in your own Web application.

1. For image upload, one line of code in the `fileUpload.asp` 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/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"
```

Note

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

2. EditLive! for XML's configuration file should now be edited to reflect the changes made in the previous step. You will find these settings within the `<httpImageUpload>` 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 `<httpImageUpload>` element if the upload script could be found at the URL `http://www.yourserver.com/scripts/fileUpload.asp`.

```
<editLive>
  ...
  <mediaSettings>
    <images>
      <httpImageUpload
        base= ...
        href="http://www.yourserver.com/scripts/fileUpload.asp"
      />
```

```
        ...
      </images>
  </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.

   ### Note

   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>
    <images>
      <httpImageUpload
        base="http://www.yourserver.com/images/"
        href="http://www.yourserver.com/scripts/fileUpload.asp"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

See Also

- Using HTTP for Image Upload in Ephox EditLive! for XML

ASP.NET HTTP Image Upload Handler Script

Summary

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.

Defining the location of the image upload handler script

The location of the image upload handler script must be defined within the XML configuration file. This setting is configured via the **href** attribute of the `<httpImageUpload>` 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! for XML the **base** attribute of the `<httpImageUpload>` element must reflect the location of the directory where images are to be stored on the Web server.

An example image upload handler script

Ephox has written a sample image upload handler script using Active Server Pages and VBScript. This script can be found at *`SDK_INSTALL`*`\webfolder\uploadscripts\aspnet\fileUpload.aspx` and *`SDK_INSTALL`*`\webfolder\uploadscripts\aspnet\fileUpload.aspx.cs` where *`SDK_INSTALL`* represents the location where the EditLive! for XML SDK is installed.

Example 10.3. Example ASP.NET Image 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 code for the `fileupload.aspx` page is as follows:

```
<%@ Page language="c#" Codebehind="fileupload.aspx.cs"
AutoEventWireup="false" Inherits="Ephox.FileUpload" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>fileupload</title>
    <meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
    <meta content="C#" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="fileupload" method="post" runat="server">
    </form>
  </body>
</HTML>
```

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! for XML to use the Image Upload Script

Below are the steps required to use the ASP.NET image upload handler from above with EditLive! for XML in your own Web application.

1. For image upload, one line of code in the `fileUpload.aspx.cs` 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/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";
```

Note

Relative paths specified within the image upload acceptor script are relative to the Web accessible location of the image upload acceptor script.

2. EditLive! for XML's configuration file should now be edited to reflect the changes made in the previous step. You will find these settings within the `<httpImageUpload>` 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 `<httpImageUpload>` element if the upload script could be found at the URL `http://www.yourserver.com/scripts/fileUpload.aspx`.

```
<editLive>
  ...
  <mediaSettings>
    <images>
      <httpImageUpload
        base= ...
        href="http://www.yourserver.com/scripts/fileUpload.aspx"
      />
        ...
      </images>
  </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.

Note

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>
    <images>
      <httpImageUpload
        base="http://www.yourserver.com/images/"
        href="http://www.yourserver.com/scripts/fileUpload.aspx"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

See Also

- Using HTTP for Image Upload in Ephox EditLive! for XML

JSP HTTP Image Upload Handler Script

Summary

This article provides a sample script, written using a Java servlet, to upload images via a HTTP POST. Instructions on how it can be tailored for use in Web applications are also included.

Defining the location of the image upload handler script

The location of the image upload handler script must be defined within the XML configuration file. This setting is configured via the **href** attribute of the `<httpImageUpload>` 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` property within the `IMAGEUPLOAD.properties` file. In order for images to function correctly within EditLive! for XML the **base** attribute of the `<httpImageUpload>` element must reflect the location of the directory where images are uploaded to on the Web server.

Setting Up the Sample JSP Image Upload Script

The JSP Image Upload Script provided with EditLive! for XML 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 XML SDK 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 where *`SDK_INSTALL`* is the directory where the EditLive! for XML SDK is installed.

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 image upload, the `IMAGEDIR` property in the `IMAGEUPLOAD.properties` file must be changed to indicate the location to which images are to be uploaded. Changing the `IMAGEUPLOAD.properties` does **not** require `UploadScript.java` to be recompiled.

```
IMAGEDIR=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 semi-colon (;) separating them. The compilation of the `UploadTest.java` file requires the Apache Commons FileUpload library, as well as the servlet library (servlet.jar).

Note

If there are spaces present in either the classpath or filename arguments then these 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"
```

Note

These locations will change dependant on your install of the EditLive! for XML 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 *`EditLive!_Java_Install`*`/WEB-INF/classes` directory (where *`EditLive_Java_Install`* represents the location that the EditLive! for XML 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 *`EditLive!_Java_Install`*`/WEB-INF/classes` directory (where *`EditLive_Java_Install`* represents the location that the EditLive! for XML SDK install used by your Web server can be found) the `web.xml` file of the application 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! for XML to Use the JSP Image Upload Script

Below are the steps required to use the JSP image upload handler with EditLive! for XML in your own Web application.

1. EditLive! for XML's configuration file should now be edited to reflect the changes made in the previous step. You will find these settings within the `<httpImageUpload>` 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 `<httpImageUpload>` 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>
    <images>
      <httpImageUpload
        base= ...
        href="http://www.yourserver.com/webapp/uploadscript"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

2. Finally the HTTP Image Upload **base** attribute should be changed to reflect the location where images can be found on your Web server.

   Note

   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>
    <images>
      <httpImageUpload
        base="http://www.yourserver.com/webapp/images/"
        href="http://www.yourserver.com/webapp/uploadscript"
      />
        ...
      </images>
  </mediaSettings>
  ...
</editLive>
```

See Also

- Using HTTP for Image Upload in Ephox EditLive! for XML

Chapter 11. Using WebDAV with EditLive! for XML

Using WebDAV with EditLive! for XML

Introduction

Ephox EditLive! for XML supports the WebDAV protocol to enable directory browsing when adding images or hyperlinks to a document. This provides the end users of EditLive! for XML with an interface to easily browse directories on the server. However, through the use of EditLive! for XML's XML configuration 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! for XML. 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! for XML

When using a WebDAV server with an instance of EditLive! for XML that has been configured accordingly results in users being able to browse the relevant WebDAV repository from within the **Insert Image** and **Insert Hyperlink** dialogs in EditLive! for XML. In the case of the **Insert Image** dialog EditLive! for XML filters the available files on the WebDAV repository according to their MIME type. This dialog will only include files which have the image/jpeg, image/gif or image/png MIME types.

Configuring EditLive! for XML for Use with WebDAV

EditLive! for XML can be easily configured for use with WebDAV via the EditLive! for XML configuration file. The configuration settings for the use of WebDAV with EditLive! for XML can be found within the **<webdav>** element of the EditLive! for XML configuration file. The **<webdav>** element contains a listing of WebDAV repositories which have their details specified by the **<repository>** elements. For more information on these elements please see the EditLive! for XML Configuration Guide.

In order to use WebDAV with EditLive! for XML then the WebDAV property or attribute of your chosen EditLive! for XML API must be set to `true`. For more information on this property or attribute please consult your EditLive! for XML API.

Basic Configuration Example

The following provides a basic example of how to configure an instance of EditLive! for XML for use with a WebDAV repository. It involves the minimum number of settings to get WebDAV functioning correctly within EditLive! for XML. The server does not implement password protection. For the purposes of this example the WebDAV server which EditLive! for XML 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! for XML (i.e. the setting of the BASE property of EditLive! for XML) is `http://www.yoursever.com/UserFiles/EditLiveFiles`. This means that the location of the WebDAV repository relative to the EditLive! for XML document base is `../WebDAV`.
- The repository should be listed to users as the `Images` repository.

The XML configuration for EditLive! for XML, in this case, would be:

```
<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 end users of EditLive! for XML to view a directory other than the root directory of the WebDAV repository by default then the **defaultDir** attribute of the **<repository>** 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 users of EditLive! for XML to access by default then the XML configuration for EditLive! for XML 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! for XML can be restricted according to the MIME of the files within the repository. As the WebDAV functionality within EditLive! for XML is used with images then files with the following MIME types will be displayed:

- image/jpeg
- image/png
- image/bmp
- image/gif

In order to activate MIME type filtering within EditLive! for XML the EditLive! for XML configuration file must contain the relevant setting. Continuing from the examples above the XML configuration for EditLive! for XML 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>
```

Note

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! for XML to use the correct username and password information. In order to do this the **<realm>**element should be used and assigned the relevant values for the **realm**, **username** and **password** for the WebDAV repository concerned. If the username and password specified are incorrect, EditLive! for XML will prompt the user for a user for a username and password when the WebDAV server is accessed.

EditLive! for XML supports the following forms of authentication:

- Basic
- Digest
- NTLM

Continuing from the examples above 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! for XML would be as follows:

```
<editLive>
  ...
  <authentication>
    <realm realm="www.yourserver.com" username="webdav" password="example" />
  </authentication>
  ...
  <mediaSettings>
    <images>
      <webdav>
        <repository
          name="Images"
          baseDir="http://www.yourserver.com/UserFiles/WebDAV"
          webDAVBaseURL="../WebDAV"
          defaultDir="images"
          useMimeType="true"
        />
      </webdav>
    </images>
  </mediaSettings>
  ...
</editLive>
```

Summary

EditLive! for XML can easily be configured to work with WebDAV repositories which exist on your Web server. The configuration of EditLive! for XML for use with WebDAV affects end users when using the **Insert Hyperlink** and **Insert Image** dialogs. In these dialogs the configuration of EditLive! for XML in this manner allows the end users to browse the server for files to link to, in the case of the **Insert Hyperlink** dialog, and, in the case of the **Insert Image** dialog, images which may be inserted.

EditLive! for XML can be configured for use with WebDAV through the **<webdav>** and **<repository>** XML elements. EditLive! for XML can be configured, through the **<realm>** XML element, to access WebDAV servers which are password protected or can be left to prompt users for a username and password.

See Also

- **<webdav>** element
- **<repository>** element
- **<realm>** element

- Enabling WebDAV on a Web Server

Enabling WebDAV on a Web Server

Introduction

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 www.webdav.org [http://www.webdav.org].

This document provides information on how WebDAV can be enabled for use on a selection of Web server and is intended as a basic guide only. If your Web server is not listed within this documentation or 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 it 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`.

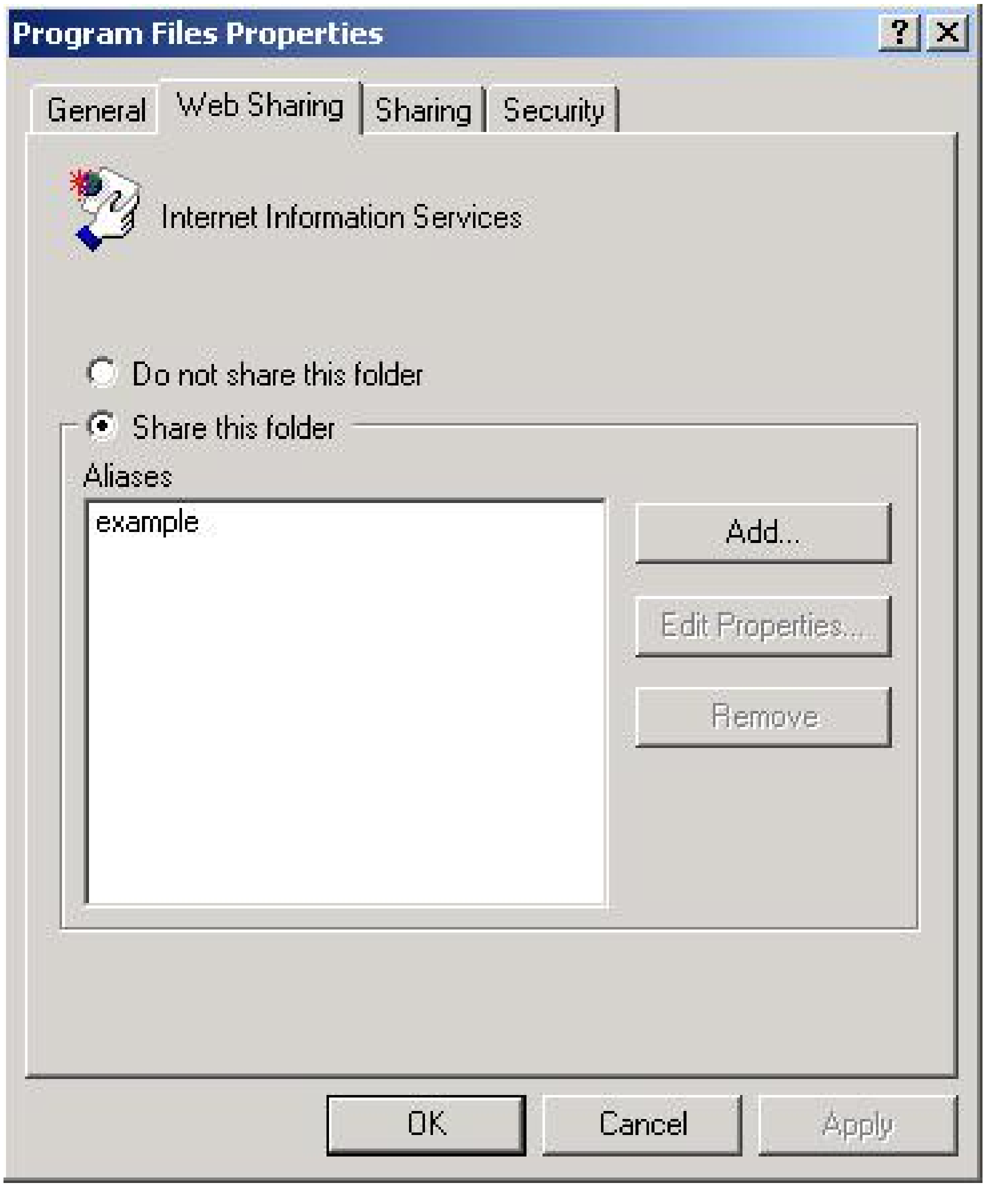
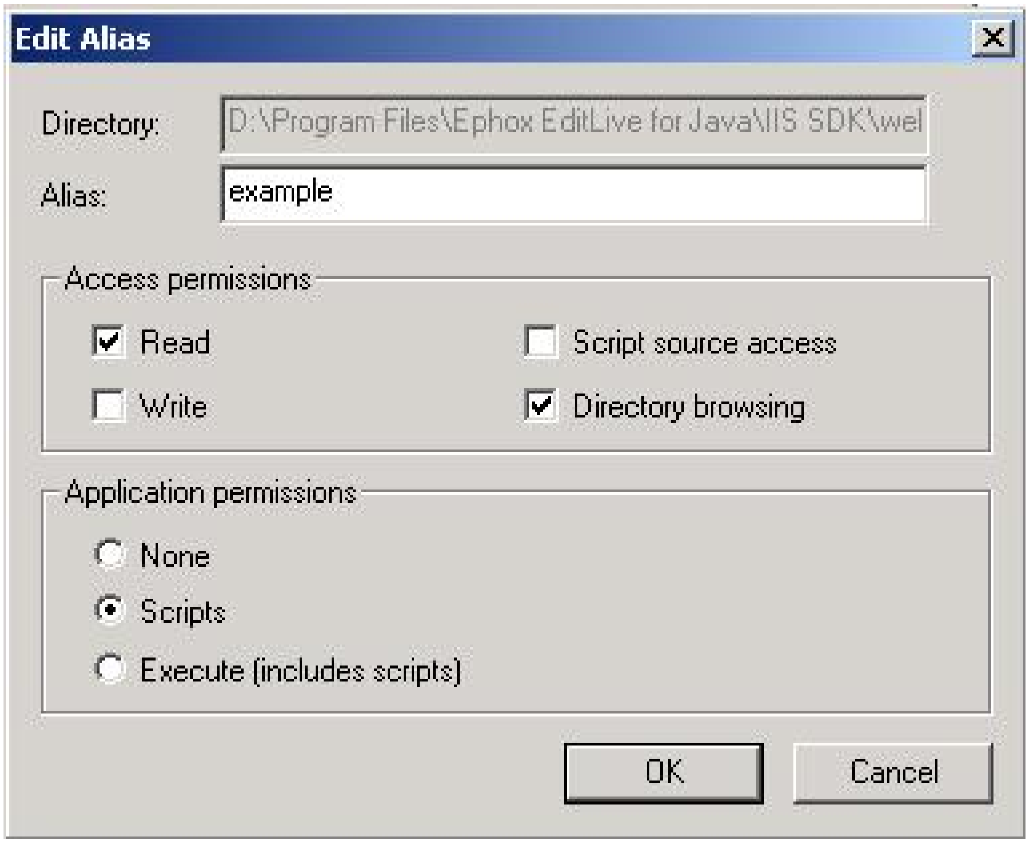


Figure 1. Folder Properties - Web Sharing

The example below demonstrates the enabling of WebDAV for the folder with the alias of `example`.



Directory: D:\Program Files\Ephox EditLive for Java\IIS SDK\wel

Alias: example

Access permissions

☑ Read

☐ Write

☐ Script source access

☑ Directory browsing

Application permissions

○ None

● Scripts

○ Execute (includes scripts)

Figure 2. Editing Web Sharing Properties

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>
-->
...
```

For more information on configuring Apache Tomcat please consult the Apache Tomcat documentation.

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 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 [http://www.webdav.org/mod_dav/install.html].

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 www.webdav.org/mod_dav/install.html#apache [http://www.webdav.org/mod_dav/install.html#apache].

Summary

A number of Web servers support the WebDAV extensions to the HTTP protocol. The processes outlined in this document should give a basic overview of how to enable the WebDAV functionality of Microsoft IIS 5.0, Apache Tomcat versions 4.0 and above and Apache Web Server with the **mod_dav** module. For more specific and detailed information on how to enable WebDAV on one of these Web servers please consult the documentation for the relevant Web server. If your Web server is not listed in this document please examine your Web server's documentation for information on whether it supports the WebDAV HTTP extensions and how to activate this functionality if it does.

EditLive! for XML provides a standard WebDAV client interface and can be used with Web servers which support the WebDAV HTTP extensions.

See Also

- Using WebDAV with EditLive! for XML
- www.webdav.org [http://www.webdav.org]
- www.webdav.org/mod_dav/ [http://www.webdav.org/mod_dav/]
- www.webdav.org/mod_dav/install.html [http://www.webdav.org/mod_dav/install.html#apache]

Chapter 12. Internationalization Support

Using Different Dictionaries with Ephox EditLive! for XML

Introduction

Ephox EditLive! for XML can be configured to use different spell checkers so that it can be used in different countries and regions. The specification of the spell checker to use with EditLive! for XML is performed via the EditLive! for XML configuration file.

Users can also add words to the local dictionary. These customizations to the EditLive! for XML dictionary are particular to a single client as the customizations are preserved on the client. To customize the dictionary on the server please refer to the article on Creating Custom Dictionaries for EditLive! for XML.

Setting the Spell Checker for EditLive! for XML

The spell checkers for EditLive! for XML for different languages are each packaged as separate files. In order to configure EditLive! for XML to use a spell checker the program must be informed of the location of the relevant spell checking package, this occurs via the EditLive! for XML configuration. The **<spellCheck>** element of the EditLive! for XML configuration provides EditLive! for XML with the location of the spell checker package which it is to use. The location is specified as a URL which can be either relative or absolute.

Dictionaries for the EditLive! for XML spell checker can be found in the `webfolder/redistributables/editlivexml/dictionaries` folder of your EditLive! for XML install. In order to use these dictionaries with EditLive! for XML the **<spellCheck>**element of the EditLive! for XML configuration should be modified to reflect the location of the dictionary to be used.

Note

The file name of the `.jar` file for the spell checker dictionary for use with EditLive! for XML must use all lower case letters.

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.

Example

The following example demonstrates how to configure EditLive! for XML to use the United States English spell checking package as its spell checker. In this example the relevant package can be found via the URL `redistributables/editlivexml/dictionaries/en_us_3_0.jar`. This example is a partial EditLive! for XML configuration document.

```
<editLive>
  ...
  <spellCheck jar="redistributables/editlivexml/dictionaries/en_us_3_0.jar" />
  ...
</editLive>
```

Available Spell Checkers

en_us_3_0.jar	English (US)
en_br_3_0.jar	English (UK)
pb_3_0.jar	Brazilian Portuguese
da_3_0.jar	Danish
du_3_0.jar	Dutch
fi_3_0.jar	Finnish
fr_3_0.jar	French - European and Canadian
ge_3_0.jar	German
it_3_0.jar	Italian
no_3_0.jar	Norwegian
po_3_0.jar	Portuguese (Iberian)
sp_3_0.jar	Spanish - European, Mexican and South American

sw_3_0.jar	Swedish

Summary

The spell checker for EditLive! for XML can be configured according to the language which you wish to use with EditLive! for XML. Each spell checker for EditLive! for XML is provided as an individual package separate from the EditLive! for XML applet package. In order to specify the spell checker to be used with EditLive! for XML the XML configuration for the relevant instance of EditLive! for XML must include a URL indicating the spell checking package in the **<spellCheck>**element.

See Also

- **<spellCheck>** XML element
- Creating Custom Dictionaries for EditLive! for XML

Using Different Character Sets with EditLive! for XML

Introduction

Ephox EditLive! for XML supports multiple character sets which allow it to be used in an international environment. The character set used by EditLive! for XML is defined within the XML document loaded into EditLive! for XML. If no character encoding is specified then EditLive! for XML will default to using UTF-8 encoding. The UTF-8 character set supports international character sets.

Supported Character Sets

EditLive! for XML supports the display and usage of the following character sets:

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	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.

Setting the Character Set via the Document

The character set to be used within EditLive! for XML can be specified in the document to be loaded into EditLive! for XML. To set the character set in this way the XML declaration at the start of the document must specify the character set of the document to be loaded into EditLive! for XML. This is done by specifying a value for the `encoding attribute`. If no character set is specified then EditLive! for XML will use UTF-8 by default.

Example 12.1. Setting the Character Set to ASCII via the XML Declaration

```
<?xml version="1.0" encoding="ASCII"?>
```

Summary

The character set for use with an instance of EditLive! for XML can be specified within the document loaded into EditLive! for XML. This declaration must be made inside the XML declaration at the beginning of the file using the `encoding` attribute. If no character set is specified then EditLive! for XML will use UTF-8 by default.

Chapter 13. Ephox CSS Extensions for Custom Tags

This chapter provides information on the Ephox cascading style sheet (CSS) extensions which may be used with the custom tag support of EditLive! for XML. Through the use of these Ephox CSS extensions developers can affect the way that custom tags are rendered in EditLive! for XML. The Ephox CSS extensions can be implemented in either an external style sheet or an embedded style sheet.

For more information on how to use Ephox CSS extensions with custom tags see the section titled Using Ephox CSS Extensions with Custom Tags.

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! for XML interprets the tag when parsing the content.

Permitted Values

block	The tag and content are displayed. The tag *must* contain content, if the tag does not contain content then it is removed. Custom block tags will not be automatically wrapped within other tags.
inline	The tag and content, if present, are displayed. The tag may or may not contain content. If the tag does not contain any content on the tag will be rendered. 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-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. If relative URLs are specified they are resolved in relation to either the BaseURL (if specified) or the page in which EditLive! for XML is embedded (if the `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 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-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. If relative URLs are specified they are resolved in relation to either the BaseURL (if specified) or the page in which EditLive! for XML is embedded (if the `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. When specifying an label to represent an empty tag this attribute must be used.

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
}
```

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 (if specified) or the page in which EditLive! for XML is embedded (if the `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-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;
}
```

Chapter 14. Using EditLive! for XML Without The Visual Designer

Creating XSLTs Without The Visual Designer

Introduction

While the Visual Designer offers a simple way for non-technical users to create forms for use within EditLive! for XML, however, developers who are familiar with XSLT may also create forms by hand to take advantage of the full power and flexibility of XSLT. EditLive! for XML has been designed with this in mind and supports most XSLTs without change. However, to take full advantage of EditLive! for XML there are some simple things to keep in mind and a number of Ephox extensions to XSLT which provide control over the additional features provided by EditLive! for XML. All the Ephox extensions to XSLT are in a separate namespace and will be ignored by standard XSLT processors. It is therefore possible to take full advantage of the features of EditLive! for XML and still use the same XSLT outside of EditLive! for XML.

Design Considerations

Context Node

The context node is a key concept in writing XSLTs and this is particularly true with EditLive! for XML. Nearly all of the EditLive! for XML's features utilize the context node in some way. When developing an XSLT for use with EditLive! for XML the XSLT should be structured so that the context node is relevant for each part of the XSLT. For instance, when creating the output for an optional element it is better to use:

```
<?xml version="1.0"?>
<xs:stylesheet version="1.0"
  xmlns:xs="http://www.w3.org/1999/XSL/Transform"
  xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled">

  <xs:template match="/">
    <html>
      <head />
      <body>
        <!-- The context node is now / -->
        <p>
        <xs:apply-templates select="/my:Untitled/my:optionalElement" />
        </p>
      </body>
    </html>
  </xs:template>

  <xs:template match="my:optionalElement">
    <!-- The context node is now my:optionalElement -->
    <xs:value-of select="." />
  </xs:template>
</xs:stylesheet>
```

as opposed to:

```
<?xml version="1.0"?>
<xs:stylesheet version="1.0"
  xmlns:xs="http://www.w3.org/1999/XSL/Transform"
  xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled">

  <xs:template match="/">
    <html>
      <head />
      <body>
        <!-- The context node is now / -->
        <p>
        <xs:value-of select="/my:Untitled/my:optionalElement" />
        </p>
      </body>
    </html>
  </xs:template>
</xs:stylesheet>
```

In a standard XSLT processor both examples produce the same output, but in EditLive! for XML the first example will automatically provide a button that allows the user to insert the optional element whereas the second example assumes the element always exists and provides a text field. For more information on how buttons are added to the EditLive! for XML interface please see the section on Automatically Added Buttons.

Imports and Includes

When including data from files other than the XML document being edited, it is important to mark the data as read only as any changes made by the user will be lost.

This occurs because only the XML document loaded into the editor is returned and thus changes to other files will be lost. The `ephox:readonly` attribute can be used to indicate that data is read only.

When specifying relative URLs for imports and includes, be aware that the XSLT or XSD must have been loaded from an URL rather than being passed into EditLive! for XML as a string so that there is a base URL to resolve it against. It is recommended that absolute URLs be used whenever possible to avoid confusion.

XPath Expressions

EditLive! for XML includes the ability to dynamically re-evaluate XPath expressions as the user makes changes to the XML document. While standard XPath expressions will work as expected in most cases there are some important considerations users of XPath should keep in mind. The dynamic XPath engine in EditLive! for XML cannot evaluate XSLT variables, therefore any XPath expressions that reference XSLT variables will be statically displayed instead of dynamically updating. XSLT designers can provide an `ephox:button` with an action of `updateView` to allow the user to re-evaluate the expression.

Using Ephox XSLT Extensions

Introduction

Ephox EditLive! for XML provides a number of extensions to XSLT to provide extra functionality and control of EditLive! for XML. These extensions are provided in a separate namespace so that they are ignored by standard XSLT processors. This means that an XSLT can be used outside of EditLive! for XML even if it contains Ephox extensions and the extensions will simply be ignored.

Declaring The Namespace

All Ephox extensions use the `http://www.ephox.com/product/editliveforxml/1.0/` namespace. Before you can use the extensions you must first declare this namespace in your XSLT. For example, to bind the Ephox extension namespace to the `ephox` prefix use the declaration below:

```
xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"
```

The normal namespace inheritance rules for XML apply. It is generally easiest to add this attribute to the `stylesheet` element of the XSLT so it is available everywhere in the document. A typical `stylesheet` element might look like:

```
<xs:stylesheet version="1.0"
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled"
 xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" xmlns="http://www.w3.org/1
```

Supported Extensions

The extensions supported by EditLive! for XML are:

`ephox:button`	A button inside the document layout that allows the user to initiate actions like a toolbar button.
`ephox:autoaddbuttons`	Controls whether or not insert and remove buttons are automatically added to the document for repeating and optional elements.
`ephox:displayas`	Specifies the type of control to use for editable values.
`ephox:displayitems`	Specifies a more user friendly set of items to display to the user in combo boxes and lists. The actual values inserted into the XML are still what is specified in the schema or in `ephox:items` but the user sees these values.
`ephox:items`	Specifies the items to provide in combo boxes and lists.
`ephox:readonly`	Specifies that a value should not be editable by the user.

Automatically Added Buttons

EditLive! for XML will automatically insert action buttons for some elements from XML. Whether action buttons are inserted for a particular element or not is dependant on both the XSD and the XSLT for the XML file being edited in EditLive! for XML. The XSD specifies whether a particular element or attribute is optional or if an element can be repeated. Action buttons will only be automatically added for optional elements or attributes or repeating elements. The insertion of action buttons is also dependant on the structure of the XSLT in use. Action buttons will only ever be inserted when the `apply-templates`, `template`, or `for-each` XSL element is used within the XSLT. The following list describes the situations in which action buttons will automatically be inserted within EditLive! for XML.

EditLive! for XML will automatically insert action buttons for the following XSL elements in the following situations:

`apply-templates` for an optional element or repeating element	EditLive! for XML will automatically insert an action button to insert the element if it doesn't already exist.
`template` for a repeating element	EditLive! for XML will automatically insert an action button to remove the element and an action button to insert another element.
`template` for an optional element or attribute	EditLive! for XML will automatically insert a remove action button for the element or attribute.
for-each for repeating elements	EditLive! for XML will automatically insert an action button to insert new elements and remove existing elements. If no elements are present an action button will be added to allow the user to add one. This is a combination of the two behaviours above.

Understanding the Current Element or Context Node

The current element for Ephox buttons is always the context node from where it appeared in the XSLT. Most operations will apply on this element so it is important to be aware of the context element, especially when using Ephox buttons (see the `ephox:button` element). In most cases, the most intuitive place to put the `ephox:button` in the XSLT is correct, however there are times when this may not be the case. As an example, to provide an action button that removes an optional element when it is present the XSLT should look like:

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:stylesheet version="1.0"
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled"
 xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"
 xmlns="http://www.w3.org/1999/xhtml">

  <xs:template match="/">
    <html>
      <head></head>
      <body>
        <xs:apply-templates select="my:document/my:optionalElement" />
      </body>
    </html>
  </xs:template>

  <!--The context node for this template is "my:optionalElement"-->
  <xs:template match="my:optionalElement" ephox:autoaddbuttons="false">
    <p>Optional value: <xs:value-of select="." />
      <ephox:button action="xmlRemove" text="Remove Optional Element" />
    </p>
  </xs:template>
</xs:stylesheet>
```

Note that a separate template is used for the `my:optionalElement` so that the context node changes to the `my:optionalElement` instead of `/`. Also note the use of `ephox:autoaddbuttons` to disable the automatically inserted remove button. The XSLT below would be incorrect and not work as intended:

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:stylesheet version="1.0"
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled"
 xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"
 xmlns="http://www.w3.org/1999/xhtml">

  <xs:template match="/">
    <html>
      <head></head>
      <body>
        <p>Optional value:
          <!--The context node here is still "/" -->
          <xs:value-of select="my:document/my:optionalElement" />
          <ephox:button action="xmlRemove" text="Remove Optional Element" />
        </p>
      </body>
    </html>
  </xs:template>
</xs:stylesheet>
```

In the second example, the context node is not changed and so the remove action button will attempt to remove the root node, resulting in an error. This XSLT will also incorrectly show the **Optional value:** label and a text box for the value of `my:optionalElement` even when it doesn't exist in the XML document. See Manually Creating XSLTs For EditLive! for XML for more information on managing the context node.

ephox:button Element

The `ephox:button` element embeds an action button in the document that, when clicked, activates a function of the editor. Typically, action buttons are used to insert, remove or move XML element in the document, however they can also be used to raise events that activate custom functionality or to re-evaluate the XSLT.

Important

Maintaining an awareness of the context node is very important when using the `ephox:button` element. For more information on how the context node is used when implementing Ephox action buttons please see the section on Understanding the Current Element or Context Node.

Note

Action buttons do not enable and disable based on XSD values, they will *always* perform their specified function. For example, an `ephox:button` element with the **action** of `xmlRemove` will always remove the relevant element when clicked. It will not disable, even when the minimum occurrences, as specified in the XSD, is reached.

Required Attributes

`action` Specifies the action to perform when the button is clicked. The value must be one of the following:

- `getCurrentNodeValue` - Get the value of the current node and supply the value, along with an `ID` for the element to the callback function. The `SetXMLNodeValue` function can then be used to set a new value for the current node.
- `xmlInsertAtCurrent` - Insert an element as a child of the current element.
- `xmlInsertAfter` - Insert an element after the current element.
- `xmlInsertBefore` - Insert an element before the current element.
- `xmlRemove` - Remove the current element.
- `xmlMoveUp` - Moves the current element before it's previous sibling.

- `xmlMoveDown` - Moves the current element after it's next sibling.
- `raiseEvent` - Raise an event back to the browser or to the embedding application.
- `updateView` - Re-evaluate the XSLT to update the view. This is useful when the output of the XSLT changes depending on the values of specific elements.

Optional Attributes

`value`	The value associated with the action. For actions that insert an XML element, the value should be the namespace and local name of the element to insert, separated by a colon. For example to insert a new `optionalElement` in the `http://www.ephox.com/product/editliveforxml/document/Untitled` namespace, the value would be:

```
http://www.ephox.com/product/editliveforxml/document/Untitled:optionalElement
```

When running as an applet, the `raiseEvent` and `getCurrentNodeValue` actions use the value as the name of the JavaScript function to call, otherwise the value is provided by the `getExtraString()` method of the `TextEvent` that is raised.

Note

The `xmlRemove`, `xmlMoveUp`, `xmlMoveDown` and `updateView` actions ignore the value attribute.

`text`	The text to display on the button.
`imageURL`	The URL to the image to use for the button. The default images used are 16 pixels by 16 pixels however the image can be any size.

ephox:autoaddbuttons Attribute

The `ephox:autoaddbuttons` attribute allows the automatically generated action buttons to be suppressed. By default, EditLive! for XML automatically detects optional and repeating elements and optional attributes then automatically inserts action buttons to allow the user to add and remove them. In some situations, the designer of the XSLT may wish to manually position these buttons or to not provide them at all and the `ephox:autoaddbuttons` attribute enables developers to prevent EditLive! for XML automatically adding action buttons.

`ephox:autoaddbuttons` attribute can be used within three XSL elements:

- `template` elements
- `apply-templates` elements
- `for-each` elements

Note

The `ephox:autoaddbuttons` attribute is only of use in XSL elements when they cause action buttons to be inserted. For more information on when EditLive! for XML will automatically add action buttons when processing XSL elements please see the section on Automatically Added Buttons.

Possible Values

The ephox:autoaddbuttons attribute requires a boolean value of either `true` or `false`.

When set to `true` EditLive! for XML will automatically provide action buttons when processing the element containing the `ephox:autoaddbuttons` attribute.

When set to `false` EditLive! for XML will not provide action buttons for the element containing the `ephox:autoaddbuttons` attribute.

Example

The example below uses the `ephox:autoaddbuttons` attribute to prevent any action buttons from being inserted for the specified element.

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:stylesheet version="1.0"
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled"
 xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"
 xmlns="http://www.w3.org/1999/xhtml">
```

```
  <xs:template match="/">
    <html>
      <head/>
      <body>
        <!-- The ephox:autoaddbuttons="false" attribute prevents an insert
             button from automatically being generated if the element
             doesn't exist in the document. -->
        <xs:apply-templates select="my:document/my:repeatingElement"
          ephox:autoaddbuttons="false"/>
      </body>
    </html>
  </xs:template>

  <!-- The ephox:autoaddbuttons="false" attribute prevents EditLive! for XML
       from automatically inserting a remove and insert after button for each
       element in the document. -->
  <xs:template match="my:repeatingElement" ephox:autoaddbuttons="false">
    <p>Value: <xs:value-of select="." /></p>
  </xs:template>
</xs:stylesheet>
```

When using `for-each` elements, the example would look like:

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:stylesheet version="1.0"
 xmlns:xs="http://www.w3.org/1999/XSL/Transform"
 xmlns:my="http://www.ephox.com/product/editliveforxml/document/Untitled"
 xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"
 xmlns="http://www.w3.org/1999/xhtml">

  <xs:template match="/">
    <html>
      <head></head>
      <body>
        <!-- The ephox:autoaddbuttons="false" attribute prevents any action
             buttons from being automatically inserted. -->
        <xs:for-each select="my:document/my:repeatingElement"
            ephox:autoaddbuttons="false">
          <p>Value: <xs:value-of select="." /></p>
        </xs:for-each>
      </body>
    </html>
  </xs:template>
</xs:stylesheet>
```

ephox:displayas Attribute

The `ephox:displayas` attribute allows the designer of the XSLT to specify an alternate control type when displaying values from the XML document. By default, EditLive! for XML automatically selects the type of control to use based on the type specified in the schema. However, in some circumstances it is useful to override the default choice and `ephox:displayas` provides a way to do so. It can only be used within `value-of` XSL elements.

Possible Values

The `ephox:displayas` attribute can only be used within `value-of` XSL elements and its value must be one of the following:

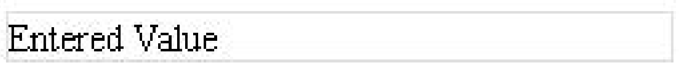
`field`	Use a standard text box. The user may enter any value without restriction.
`password`	A password text box is used. The user may enter any value without restriction, however the current value is shown only as bullets rather than showing the actual characters. This is suitable for allowing the user to enter a password while preventing by-standers from seeing the password. Note that the actual value is inserted into the XML document *without* any encryption and will be visible when submitted or if the user switches to code view. When security is a concern, HTTPS should be used to submit the document.
`checkbox`	A checkbox is used to allow the user to specify either `true` or `false`. If the checkbox is checked, `true` will be inserted into the XML file, otherwise `false` will be inserted. It is not currently possible to specify different values to insert - in these cases a combo box or list should be used instead.
`combo`	A combo box is used and the user must select one of the options from the combo box. The items in the combo box are specified with the `ephox:items` attribute and the `ephox:displayitems` attribute.
`editableCombo`	Uses a combo box that will allow the user to select an item from a drop down or type in their own value. The items in the drop down are specified with the `ephox:items` attribute and the `ephox:displayitems` attribute.
`list`	A list is used and the user must select one of the values in the list.

	A scroll bar is provided if the number of list items exceeds the available space. The items in the list are specifed with the `ephox:items` attribute and the `ephox:displayitems` attribute.
`date`	A date picker is provided for the user to select from. The value inserted into the XML is compatible with the W3C schema `date` type.
`time`	A time picker is used.
`dateTime`	A combination control that allows the user to specify and date and time value is provided.

Examples

Using a Standard Field

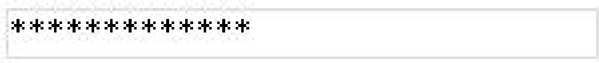
```
<xs:value-of select="." ephox:displayas="field"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0"/>
```



Entered Value

Using a Password Field

```
<xs:value-of select="." ephox:displayas="password"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



****

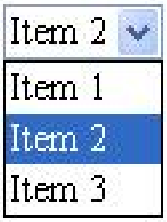
Using a Check Box

```
<xs:value-of select="." ephox:displayas="checkbox"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```

☑

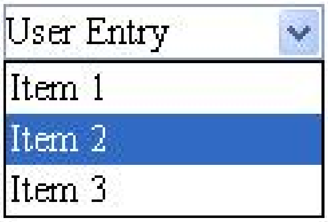
Using a Combo Box

```
<xs:value-of select="." ephox:displayas="combo"
  ephox:items="item1,item2,item3" ephox:displayitems="Item 1,Item 2,Item 3"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



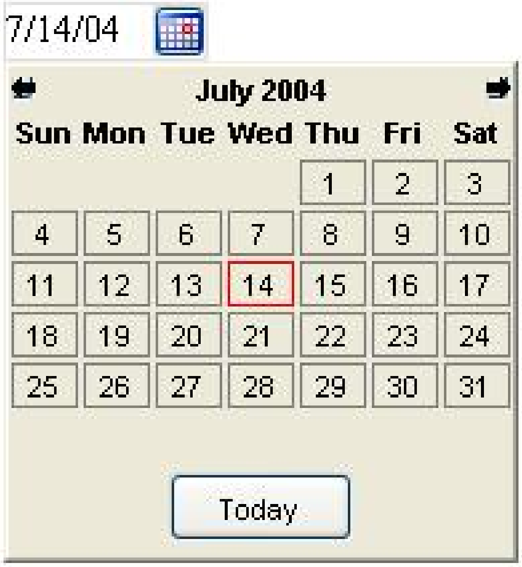
Using an Editable Combo Box

```
<xs:value-of select="." ephox:displayas="editableCombo"
  ephox:items="item1,item2,item3" ephox:displayitems="Item 1,Item 2,Item 3"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



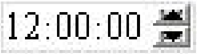
Using a Date Picker

```
<xs:value-of select="." ephox:displayas="date"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



Using a Time Picker

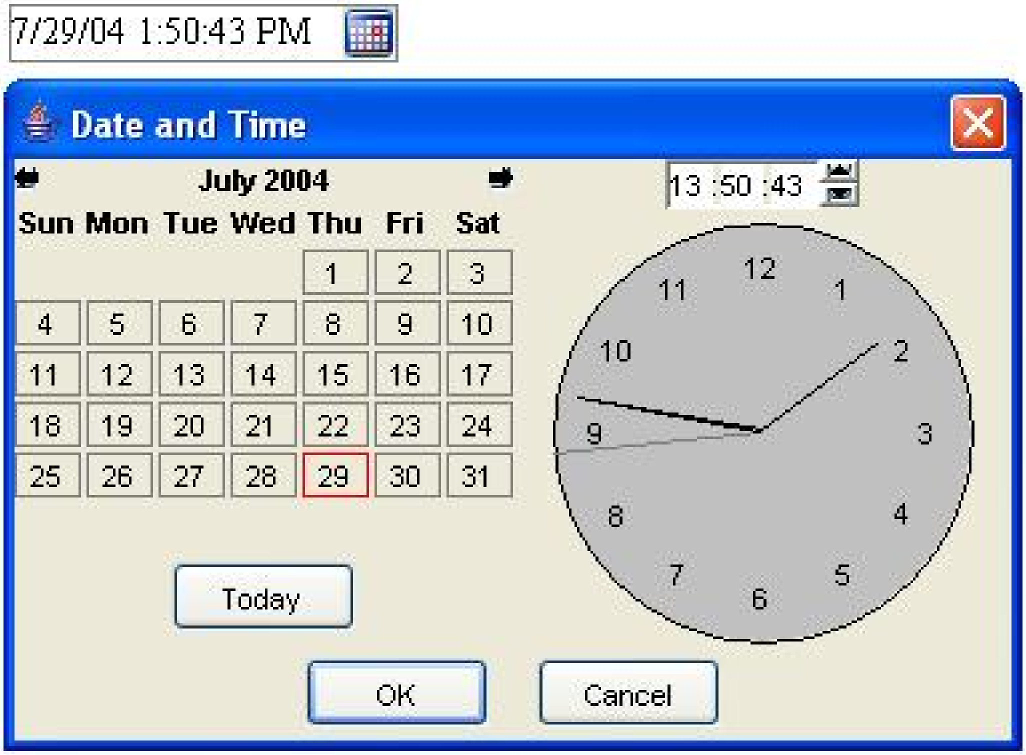
```
<xs:value-of select="." ephox:displayas="time"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



12:00:00

Using a Date and Time Picker

```
<xs:value-of select="." ephox:displayas="dateTime"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```



7/29/04 1:50:43 PM

13 :50 :43

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

ephox:items and ephox:displayitems Attributes

Introduction

The `ephox:items` and `ephox:displayitems` attributes are used together to specify the items to display in combo boxes, editable combo boxes and lists. `ephox:displayitems` is used to specify the items to display to the user and `ephox:items` is used to specify the corresponding values to insert into the XML. The `ephox:items` and `ephox:displayitems` attributes should only be used within XSL `value-of` elements.

ephox:items Attribute

The `ephox:items` attribute is used to specify a list of values for combo boxes, editable combo boxes and lists. The value is a comma separated list of items. When ephox:items is used without `ephox:displayitems` each item is displayed to the user, and when selected inserted into the XML. However when `ephox:displayitems` is used, the items in `ephox:displayitems` are displayed to the user and when selected, the corresponding item in `ephox:items` is inserted into the XML. The `ephox:items` attribute should only be used within XSL `value-of` elements.

When `ephox:items` is not specified, the values for combo boxes, editable combo boxes and lists are taken from the schema for the document.

Possible Values

The value of the `ephox:items` attribute must be a comma separated list of items. This list represents the list of values which may be inserted into the XML document for the relevant element or attribute. When using the `ephox:items` attribute in tandem with the `ephox:displayitems` attribute the number of items in the comma separated lists for each attribute must match.

ephox:displayitems Attribute

The `ephox:displayitems` attribute is used to specify a list of items to display in combo boxes, editable combo boxes and lists. The value is a comma separated list of items to display. `ephox:displayitems` must be used with either `ephox:items` or a schema type that has enumeration facets specifying allowed values. Each item in `ephox:displayitems` corresponds in order to an item in `ephox:items` or the schema enumerations and is used as a user friendly version of the value. There must be exactly the same number of items in ephox:displayitems as there are specified values. The `ephox:displayitems` attribute should only be used within XSL `value-of` elements.

Possible Values

The value of the `ephox:displayitems` attribute must be a comma separated list of items. This list represents the list of values which are displayed for the relevant element or attribute in the XML document. When using the `ephox:displayitems` attribute in tandem with the `ephox:items` attribute the number of items in the comma separated lists for each attribute must match.

Examples

Using `ephox:items` And `ephox:displayitems` Together

The example below will display as a combo box with the items:

- `Item 1`
- `Item 2`
- `Item 3`

When the user selects an item, the value from `ephox:items` will be inserted. For example if `Item 1` is selected by the user `item1` will be inserted into the XML. Similarly if the user selects `Item 3`, `item3` will be inserted into the XML.

```
<xs:value-of select="." ephox:displayas="combo"
  ephox:items="item1,item2,item3" ephox:displayitems="Item 1,Item 2,Item 3"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```

Using `ephox:items` Alone

The example below will display as a list with the items:

- `item1`
- `item2`
- `item3`

When the user selects an item it will be inserted into the XML exactly as it is displayed.

```
<xs:value-of select="." ephox:displayas="combo"
  ephox:items="item1,item2,item3"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```

Using ephox:displayitems Alone

`ephox:displayitems` can be used without `ephox:items` if the type in the schema uses enumeration facets to specify a list of allowed values. In this case, `ephox:displayitems` provides a user friendly form for each enumerated value. If the schema declares an element as:

```
<xsd:element name="favoriteEditor">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="elx" />
      <xsd:enumeration value="elj" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The XSLT designer may wish to provide a more intuitive list of items for the user to select from. The XSLT snippet will achieve this.

```
<xs:value-of select="my:favoriteEditor"
  ephox:displayitems="EditLive! for XML,EditLive! for Java"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```

ephox:readonly Attribute

Introduction

The `ephox:readonly` attribute allows values to be displayed without making them editable by the user. The value appears in the output as plain text exactly as it would if run through a standard XSLT processor. The `ephox:readonly` attribute can only be used on `value-of` elements.

By default elements from an XML document rendered in EditLive! for XML are editable.

Possible Values

The `ephox:readonly` attribute requires a boolean value of either `true` or `false`.

When set to `true` EditLive! for XML will display the value of the relevant element as plain text which will *not* be editable.

When set to `false` EditLive! for XML will display the value of the relevant element from the XML as an editable field.

Example

The example below inserts the value of the context node into the display without making it editable.

```
<xs:value-of select="." ephox:readonly="true"
  xmlns:ephox="http://www.ephox.com/product/editliveforxml/1.0" />
```

Chapter 15. Instantiation API

This section of the API guide includes methods and properties which may be used when instantiating the EditLive! for XML or Visual Designer applets . These method and properties may only be accessed prior to calling the **show** method.

EditLive! for XML JavaScript Constructor

Description

This method creates an instance of an Ephox EditLive! for XML Javascript object.

Note

This method only applies for EditLive! for XML JavaScript integrations.

Syntax

JavaScript

```
new EditLiveXML(strName, intWidth, intHeight);
```

Parameters

strName	A unique string identifier for this instance of EditLive! for XML. This is a required parameter.
intWidth	An integer specifying the width of the applet when displayed. This is a required parameter.
intWidth	An integer specifying the height of the applet when displayed. This is a required parameter.

Examples

Example 15.1. JavaScript EditLive! Constructor Scripting Example

The following code creates an EditLive! for XML object and assigns the identifier `editlivejs` 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;
editlivejs = new EditLiveXML("ELApplet1","700","400");
```

Remarks

The constructor must be called before any operations can occur on the EditLive! for XML object.

Ephox recommends setting the width and height in pixels, as on Macintosh machines, if these values are set as percentages and the Web browser window is resized, EditLive! for XML will not be resized with the window.

Visual Designer JavaScript Constructor

Description

This method creates an instance of the Visual Designer Javascript object.

Note

This method only applies for the Visual Designer JavaScript integrations.

Syntax

JavaScript

```
  new VisualDesigner(strName, intWidth, intHeight);
```

Parameters

strName	A unique string identifier for this instance of the Visual Designer.

	This is a required parameter.
intWidth	An integer specifying the width of the applet when displayed. This is a required parameter.
intWidth	An integer specifying the height of the applet when displayed. This is a required parameter.

Examples

Example 15.2. JavaScriptt Visual Designer Constructor Scripting Example

The following code creates a Visual Designer object and assigns the identifier `designer` to the JavaScript variable. The object has a unique name of `VDApplet1`, a width of `700` pixels and a height of `400` pixels.

```
var designer;
designer = new VisualDesigner("VDApplet1","700","400");
```

Remarks

The constructor must be called before any operations can occur on the Visual Designer object.

Ephox recommends setting the width and height in pixels, as on Macintosh machines, if these values are set as percentages and the Web browser window is resized, the Visual Designer will not be resized with the window.

addView Method

Description

This method is used to add a view to EditLive! for XML. A view is presented to the user as a tab within EditLive! for XML. This method can be called multiple times in order to add multiple views to EditLive! for XML.

Note

This method cannot be used with the Visual Designer. To set a view for the Visual Designer the **addViewAsText** method must be used.

Syntax

JavaScript

```
addView(strName,strViewURL);
```

Parameters

strName	A string specifying the name for the view being added. This name will be displayed on the tab representing the view on the interface of EditLive! for XML.
strViewURL	A URL specifying the Web accessible location of the file for this view. This value can be either a relative or an absolute URL. Relative URLs are relative to the location of the page in which EditLive! for XML is embedded.

Examples

Example 15.3. addView Method Example Scripting

The following example demonstrates how to load the views specified by the URLs `http://yourserver.com/views/viewOne.xsl` and `../views/secondView.xsl`. These views are named `First View` and `Second View` respectively.

JavaScript

```
var editlivexml = New EditLiveXML("ELXApplet1","700","400");
editlivexml.addView("First View","http://yourserver.com/views/viewOne.xsl");
editlivexml.addView("Second View","../views/secondView.xsl");
```

Remarks

This method can be called multiple times to add multiple views to EditLive! for XML.

Each view added to EditLive! for XML is presented to a user via a tab on the interface. The tab for the view will be labeled with the value of the **strName** parameter.

Views can also be added as a string of text via the **addViewAsText** method.

addViewAsText Method

Description

This method allows a view to be specified as a string of text. The string provided to this method as a parameter must contain a complete XML stylesheet document for use as a view within EditLive! for XML or the Visual Designer. The name parameter supplied with the view provides the name for view which is used to represent the view within the user interface.

Loading views via the **addViewAsText** method can reduce the load time of EditLive! for XML. This can be most easily achieved by using server-side scripting to load the view file from the server's file system into a scripting string variable which can then be used when instantiating EditLive! for XML.

Note

When instantiating the Visual Designer this method should be used to add views to the Visual Designer.

Syntax

JavaScript

```
addViewAsText(strName,strViewText);
```

Parameters

strName	A string specifying the name for the view being added. In EditLive! for XML the name will be displayed on the tab representing the view on the interface.

	In the Visual Designer the name will be displayed within the list of views available within the designer.
strXMLText	A string which contains the text of the view (XSL) document to be added to this instance of EditLive! for XML.

Examples

Example 15.4. addViewAsText Method Example Scripting

The following code would specify that Ephox EditLive! for XML is to include the given view, the view will be listed with the name `View One`.

JavaScript

```
editlive.addViewAsText("View One", "<xs:stylesheet xmlns:xs=...");
```

Note

The view string parameter for this method must be URL encoded. It is recommended that a server-side URL encoding function be used if available. The usage of the JavaScript **escape** function is *not* recommended as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Note

The XSL document in the examples above is incomplete, and will not function. It is given only as a example to aid understanding. The XSL document passed to EditLive! for XML via this method should not contain carriage return or new line characters.

Remarks

The view string parameter for this method must be URL encoded. It is recommended that a server-side URL encoding function be used if available. The usage of the JavaScript **escape** function is *not* recommended as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Loading views via the **addViewAsText** method can reduce the load time of EditLive! for XML. This can be most easily achieved by using server-side scripting to load the view file from the server's file system into a scripting string variable which can then be used when instantiating EditLive! for XML.

addXSDAsText Method

Description

This method specifies an XML Schema Document to be used with an instance of EditLive! for XML. This method can be called multiple times in order to supply EditLive! for XML with several XSDs. This method accepts an XSD as a text string. Loading XSDs via the **addXSDAsText** method can reduce the load time of EditLive! for XML. This can be most easily achieved by using server-side scripting to load the XSD file from the server's file system into a scripting string variable which can then be used when instantiating EditLive! for XML.

Note

This property cannot be used with the Visual Designer. To set the XSD for use with the Visual Designer the **XSDAsText** property must be used.

Syntax

JavaScript

```
addXSDAsText(strXSD);
```

Parameters

strXSD	A string which contains a full XSD to be used with this instance of EditLive! for XML.

Note

The XSD string should be URL encoded. This can be achieved with a server side URL encoding method.

Examples

Example 15.5. addXSDAsText Method Example Scripting

The following example adds an XSD for use with EditLive! for XML.

JavaScript

```
editlivejs.addXSDAsText = "<xsd:schema targetNamespace=...";
```

Note

The view string parameter for this method must be URL encoded. It is recommended that a server-side URL encoding function be used if available. The usage of the JavaScript **escape** function is *not* recommended as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Note

The XSD in the examples above is incomplete, and will not function. It is given only as a example to aid understanding. The XSD passed to EditLive! for XML via this method should not contain carriage return or new line characters.

Remarks

Using this method instead of setting the **setXSDURL** property may result in faster load times for EditLive! for XML.

The string parameter for this method must be URL encoded. It is recommended that a server-side URL encoding function be used if available. The usage of the JavaScript **escape** function is *not* recommended as the JavaScript **escape** function does not fully comply with the URL encoding standard.

show Method

Description

This method displays the Ephox EditLive! for XML instance object upon which it is called in the Web browser.

Syntax

JavaScript

```
show();
```

Examples

Example 15.6. show Method Example Scripting

The following code sets only the required properties of an EditLive! for XML applet and then displays the applet within the Web page.

JavaScript

```
var editlive_js;
editlive_js = new EditLiveXML("ELApplet1","700","400");
editlive_js.setDownloadDirectory("../../redistributables/editlivexml");
editlive_js.setConfigurationFile("sample_elconfig.xml");
editlive_js.setDocument(escape("<p>Document body contents</p>"));
editlive_js.show();
```

Remarks

Before this method is called, all the required properties of an EditLive! for XML instance object must be set.

See Also

- ShowAsButton Method

showAsButton Method

Description

This method displays the Ephox EditLive! for XML instance object in the Web browser as a button. When passed the parameter `true` EditLive! for XML will automatically open in a popout window, clicking an associated button in the browser will hide the window. When set to `false` EditLive! for XML displays as a button which must be clicked in order for the popout window containing EditLive! for XML to appear.

Syntax

JavaScript

```
show(blnPopout);
```

Parameters

blnPopout	A boolean indicating whether EditLive! for XML should automatically open in a popout window. When set to `true` EditLive! for XML will automatically open in a popout window, with an associated button in the browser, clicking the button will hide the window. When set to `false` EditLive! for XML displays as a button which must be clicked in order for the popout window containing EditLive! for XML to appear.

Examples

Example 15.7. showAsButton Method Example Scripting

The following example demonstrates how to cause EditLive! for XML to automatically appear in a popout window which has an associated button in the browser for showing and hiding the window.

JavaScript

```
var editlive_js;
editlive_js = new EditLiveXML("ELApplet1","700","400");
editlive_js.setDownloadDirectory("../../redistributables/editlivexml");
editlive_js.setConfigurationFile("sample_elconfig.xml");
editlive_js.setDocument(escape("<p>Document body contents</p>"));
editlive_js.showAsButton("true");
```

Remarks

Before this method is called, all the required properties of an EditLive! for XML instance object must be set.

When using this method it must be called instead of the **show** method.

To set the display icons and text for the button created through the use of this method the following properties can be set:

- ShowButtonIconURL
- ShowButtonText
- HideButtonIconURL
- HideButtonText

AutoSubmit Property

Description

This property specifies the way in which Ephox EditLive! for XML behaves when the page is submitted. This affects how content is retrieved from EditLive! for XML.

Syntax

JavaScript

```
setAutoSubmit(blnSubmit);
```

Parameters

blnSubmit	A boolean indicating if EditLive! for XML should attach its content submission to the **onsubmit** function. The default value is `true`.

Example

Example 15.8. AutoSubmit Property Example Scripting

The following code would inform EditLive! for XML to **not** attach its content submission to the **onsubmit**function.

JavaScript

```
editlivejs.setAutoSubmit(false);
```

Remarks

When attaching its content submission to the **onsubmit** function EditLive! for XML 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! for XML instance and is given the name that was specified by the developer when the EditLive! for XML instance was created. For example, if the applet was assigned the ELApplet1 was specified in the above example so 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! for XML 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! for XML 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! for XML by setting the **AutoSubmit** property to `false` the developer may wish to retrieve content from EditLive! for XML using the **GetDocument** function provided in the EditLive! for XML JavaScript Run Time API.

BaseURL Property

Description

This property can be used to set the base URL used by EditLive! for XML 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 configuration file. For example, when a single XML configuration file is used within a system where EditLive! for XML is used in multiple instances for editing pages with differing base URLs.

Syntax

JavaScript

```
setBaseURL(strBaseURL);
```

Parameters

strBaseURL	A string specifying the base URL to be used with this instance of EditLive! for XML. The URL should map to a virtual directory. The base URL is used by EditLive! for XML when resolving any relative URLs.

Example

Example 15.9. BaseURL Property Example Scripting

The following code would set the base URL for an instance of EditLive! for XML to `http://www.yourserver.com/editor/`. This URL will be used when resolving all relative URLs in the EditLive! for XML content and configuration file (e.g. URLs for images and links).

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 however `http://www.yourserver.com/editor` is not.

Any value set in the <base> element of the EditLive! for XML configuration 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 configuration file.

See Also

- <base> element

ConfigurationFile Property

Description

This property or the **ConfigurationText** property (but not both) must be set for an Ephox EditLive! for XML applet to run.

This property specifies the URL at which the XML configuration file for EditLive! for XML can be found. This file will customize the EditLive! for XML interface. You may like to look at the ConfigurationText property if you are considering dynamically generating XML configuration files, but otherwise it is probably simpler to use the ConfigurationFile property.

Syntax

JavaScript

```
setConfigurationFile(strFileURL);
```

Parameters

strFileURL	A string which is the URL for where the XML configuration file for this instance of EditLive! for XML can be requested from.

Examples

Example 15.10. ConfigurationFile Property Example Scripting

The following code would specify that EditLive! for XML is to load with the properties as specified by the file config.xml which can be found at `http://someserver/xmlconfig/config.xml`.

JavaScript

```
editlivejs.setConfigurationFile("http://somesever/xmlconfig/config.xml");
```

Remarks

Using the **ConfigurationText** property to configure EditLive! for XML results in a faster load time for the EditLive! for XML applet than is achieved through the use of the **ConfigurationFile** property.

The **ConfigurationFile** property is mutually exclusive with the ConfigurationText property. You should provide either a URL to a configuration file, or pass in the configuration text in a String format.

See Also

- ConfigurationText property

ConfigurationText Property

Description

This property or the **ConfigurationFile** property (but not both) is required to be set for an Ephox EditLive! for XML applet to run.

This property specifies the XML configuration text to be used by Ephox EditLive! for XML. This text will customise the Ephox EditLive! for XML interface. To find out about how to use XML Configuration files, please read EditLive! for XML Configuration reference. The **ConfigurationText** property allows the configuration of EditLive! for XML to be via a string which contains the configuration XML document to be used with EditLive! for XML. Loading the configuration for EditLive! for XML via the **ConfigurationText** property can reduce the load time of EditLive! for XML. 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! for XML.

Syntax

JavaScript

```
setConfigurationText(strXMLText);
```

Parameters

strXMLText	A string which contains the text of the XML configuration document for this instance of Ephox EditLive! for XML.

Examples

Example 15.11. ConfigurationText Property Example Scripting

The following code would specify that Ephox EditLive! for XML is to load with the given XML Configuration Text.

JavaScript

```
editlivejs.setConfigurationText('%3C%3Fxml%20version%3D%221.0%22%3F%3E...');
```

Note

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 **escape** function does not fully comply with the URL encoding standard.

Note

The XML document in the examples above URL encoded and incomplete. It is given only as an example to aid understanding. The XML document passed to EditLive! for XML via this method should by URL encoded.

Remarks

Using the **ConfigurationText** property to configure EditLive! for XML results in a faster load time for the EditLive! for XML applet than is achieved through the use of the **ConfigurationFile** property.

The **ConfigurationText** property is mutually exclusive with the ConfigurationFile property. You should provide either a URL to a configuration file, or pass in the configuration text in a String format.

When using the JavaScript **setConfigurationText** property the string parameter must be URL encoded. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

See Also

- ConfigurationFile property

Cookie Property

Description

This property stipulates the name of the cookie to be used by Ephox EditLive! for XML.

Syntax

JavaScript

```
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

Example 15.12. Cookie Property Example Scripting

The following code would set the **Cookie** property to "`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! for XML.

DebugLevel Property

Description

This property stipulates the level of debugging to be used when running EditLive! for XML.

Syntax

JavaScript

```
  setDebugLevel(strDebug);
```

Parameters

strDebugLevel	A string specifying the level of debugging to run EditLive! for XML with. There are several distinct possible debug levels:

- `fatal`

- `error`
- `warn`
- `info`
- `debug`
- `http`

The default value is `info`.

Examples

Example 15.13. DebugLevel Property Example Scripting

The following code would specify that the debug level is set to `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! for XML from continuing, thus resulting in termination of the program.
error	This debugging level displays error messages for cases in which EditLive! for XML can continue despite the error. However, the current EditLive! for XML operation will most likely fail due to the relevant error. This debugging level also displays all the debugging 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! for XML to behave in an unexpected manner. However, the current EditLive! for XML operation will most likely be completed successfully. This debugging level also displays all the debugging 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! for XML expects such events and deals with them accordingly. This debugging level also displays all the debugging 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 debugging 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! for XML (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 debugging information that would be displayed should the debugging level be set to `debug`.

Document Property

Description

This property specifies the initial document contents of the Ephox EditLive! for XML applet.

Note

This property cannot be used with the Visual Designer.

Syntax

JavaScript

```
setDocument(strDocument);
```

Parameters

strDocument	A string specifying the initial document contents of the EditLive! for XML applet. The default value is an empty string.

Example

Example 15.14. Document Property Example Scripting

The following code would set the initial document contents of EditLive! for XML to be equal to "`Initial contents of Ephox EditLive!`".

JavaScript

```
editlivejs.setDocument(escape("<HTML><HEAD><TITLE>Example</TITLE></HEAD>
<BODY><P>Initial contents of Ephox EditLive!</P></BODY></HTML>"));
```

Note

The string passed to the JavaScript **setDocument** property must be URL encoded or encoded using the JavaScript **escape** function. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Remarks

When using the JavaScript **setDocument** property the string parameter must be URL encoded or encoded using the JavaScript **escape** function. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

DownloadDirectory Property

Description

This property must be set for all EditLive! for XML global objects.

This property specifies the directory in which the Ephox EditLive! for XML source files can be found on the server.

Syntax

JavaScript

```
setDownloadDirectory(strDownloadDirectory);
```

Parameters

strDownloadDirectory	A string specifying the location of the EditLive! for XML source files and JavaScript library.

Examples

Example 15.15. DownloadDirectory Property Example Scripting

The following code would specify that the source files were in a directory called `redistributables/editlivexml` on the Web server.

JavaScript

```
editlivejs.setDownloadDirectory("../../redistributables/editlivexml");
```

Remarks

This property must be set when instantiating EditLive! for XML

HideButtonIconURL Property

Description

This property specifies the URL at which the icon to be used with the *hide* button for EditLive! for XML. This function can only be used when displaying EditLive! for XML as a button in the browser using the **ShowAsButton** method.

Syntax

JavaScript

```
setHideButtonIconURL(strImageURL);
```

Parameters

strImageURL	A string which is the URL for where the icon image for the *hide* button for this instance of EditLive! for XML can be requested from.

Examples

Example 15.16. HideButtonIconURL Property Example Scripting

The following code would specify that EditLive! for XML is to use the image `http://server.com/icons/hideButton.gif` when displaying the *hide* button for EditLive! for XML.

JavaScript

```
editlivejs.setHideButtonIconURL("http://server.com/icons/hideButton.gif");
...
editlivejs.showAsButton("true");
```

Remarks

This function should only be used when the **ShowAsButton** method is used to display EditLive! for XML.

See Also

- ShowAsButton method
- ShowButtonText property

- ShowButtonIconURL property
- HideButtonText property

HideButtonText Property

Description

This property specifies the text to be used with the *hide* button for EditLive! for XML. This function can only be used when displaying EditLive! for XML as a button in the browser using the **ShowAsButton** method.

Syntax

JavaScript

```
setHideButtonText(strFileURL);
```

Parameters

strButtonText	A string which is the text to be displayed on the *hide* button for this instance of EditLive! for XML when it is displayed as a button.

Examples

Example 15.17. ShowButtonIconURL Property Example Scripting

The following code would specify that EditLive! for XML is to use the text `Hide EditLive! for XML` when displaying the *hide* button for EditLive! for XML.

JavaScript

```
editlivejs.setHideButtonText(escape("Hide EditLive! for XML"));
...
editlivejs.showAsButton("true");
```

Note

The string passed to the JavaScript **setHideButtonText** property must be URL encoded. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Remarks

This function should only be used when the **ShowAsButton** method is used to display EditLive! for XML.

The string passed to the JavaScript **setHideButtonText** property must be URL encoded. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

See Also

- ShowAsButton method
- ShowButtonText property
- ShowButtonIconURL property
- HideButtonIconURL property

JREDownloadURL Property

Description

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! for XML.

Important

When deploying the JRE from a location other than Sun Microsystems' servers this property must be set.

Syntax

JavaScript

```
setJREDownloadURL(strJREURL);
```

Parameters

strJREURL	A string containing the location to download the Java Runtime Environment from if it is required to be installed.

Examples

Example 15.18. JREDownloadURL Property Example Scripting

The following sets the JRE download URL to be `../JREDownload/j2re-1_4_1-windows-i586-i.exe`.

JavaScript

```
editlivejs.setJREDownloadURL("../JREDownload/j2re-1_4_1-windows-i586-i.exe");
editlivejs.setLocalDeployment(true);
editlivejs.setMinimumJREVersion("1.4.0");
```

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! for XML applet is embedded.

The **LocalDeployment** property must be set to `true`.

The **MinimumJREVersion** property must be set to a JRE version which has a version number less than or equal to that of the JRE found at the location specified in the **JREDownloadURL** property.

The **JREDownloadURL** property must specify the URL of a JRE installer executable.

See Also

- **LocalDeployment** Property
- **MinimumJREVersion** Property

LocalDeployment Property

Description

This property stipulates where to download the Java Run Time Environment (JRE) from, if it is not already installed on the client machine. If this property is set to `true`, the JRE will be downloaded and installed from the **server** directory stipulated by the **JREDownloadURL** property, if required on the client machine. If this property is set to `false`, the JRE will be downloaded and installed from the Sun Microsystems [http://www.sun.com] Web site, if required on the client machine.

Important

When setting this property to `true` it should be ensured that the **JREDownloadURL** is set to the location of the JRE installer with the the JRE will be deployed.

Syntax

JavaScript

```
setLocalDeployment(blnLocalDeployment);
```

Parameters

blnLocalDeployment	A boolean value indicating if the JRE should be deployed from Sun Microsystems' servers if it is not already present. When set to `false` the JRE will be deployed from Sun Microsystems' servers. The default value is `false`.

Example

Example 15.19. LocalDeployment Property

The following code would set the **LocalDeployment** property to `true`.

JavaScript

```
editlivejs.setLocalDeployment(true);
```

See Also

- DownloadDirectory property

Locale Property

Description

Explicitly sets the locale that EditLive! for XML should use for the interface translation, date formats and other locale dependant properties. If this property is not set the locale for EditLive! for XML 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! for XML the English translation of the user interface is used by default.

Valid locales for EditLive! for XML include:

- EN - English
- ES - Spanish
- DE - German
- IT - Italian
- KO - Korean
- FR - French

- CS - Czech
- ZH - Simplified Chinese
- DA - Danish

Syntax

JavaScript

```
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

Example 15.20. Locale Property Example Scripting

The following code would set the locale of EditLive! for XML to German. This means that EditLive! for XML will run with the German interface translation.

JavaScript

```
editlivejs.setLocale("DE");
```

Remarks

If the locale is not set then the locale used by the client will be as specified by their system properties. If the locale of the client machine is not supported EditLive! for

XML will default to the English interface translation.

If the specified locale is not supported EditLive! for XML will default to the English interface translation.

The EditLive! for XML interface has been translated into the following languages for the listed locales:

- EN - English
- ES - Spanish
- DE - German
- IT - Italian
- KO - Korean
- FR - French
- CS - Czech
- DA - Danish
- ZH - Simplified Chinese
- AR - Arabic

MinimumJREVersion Property

Description

This attribute specifies the minimum version of the Java Runtime Environment required to run EditLive! for XML. It should be noted that EditLive! for XML should be used with JRE version 1.4.0 and above.

Syntax

JavaScript

```
setMinimumJREVersion(strJREVersion);
```

Parameters

strJREVersion	A string defining the version number of the minimum JRE version which EditLive! for XML is to run with. The default value is `1.4.0`.

Examples

Example 15.21. MinimumJREVersion Property Example Scripting

The following sets the minimum JRE version to be 1.4.1

JavaScript

```
editlivejs.setMinimumJREVersion("1.4.1");
```

Remarks

Currently supported JRE versions are 1.4.0 and above.

Care should be taken when using this property in conjunction with the **JREDownloadURL** property. See the **JREDownloadURL** property for more information.

See Also

- **JREDownloadURL** Property

OnInitComplete Property

Description

This property can be used to call a JavaScript function once EditLive! for XML has finished loading. Once EditLive! for XML has finished loading the JavaScript function defined by the **OnInitComplete** property is used as a callback.

Syntax

JavaScript

```
setOnInitComplete(strCallBack);
```

Parameters

strCallBack	A string specifying the name of a JavaScript function to use as the callback function once EditLive! for XML has finished loading.

Example

Example 15.22. OnInitComplete Property JavaScript Callback Function Example

The following code provides the JavaScript callback function which will set the body content of EditLive! for XML once the applet has finished loading. The callback function is named `AppletLoaded`. The content of EditLive! for XML will be set to `Body content set at runtime`. The name of the EditLive! for XML Applet JavaScript object is `ELApplet1_js`.

Note

The `AppletLoaded` function uses the JavaScript runtime API to set the body content of EditLive! for XML. The string used to set the body content in this example is URL encoded.

```
<script language="javascript">
  function AppletLoaded(){
    ELApplet1_js.SetDocument("%3Cp%3ESet+content+at+runtime%3C%2Fp%3E");
  }
</script>
```

Example 15.23. OnInitComplete Property Example Scripting

The following example instantiate a version of EditLive! for XML and assign 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.

JavaScript

```
var ELApplet1_js;
ELApplet1_js = new EditLiveXML("ELApplet1","700","400");
ELApplet1_js.setConfigurationFile("sample_elconfig.xml");
ELApplet1_js.setOnInitComplete("AppletLoaded");
```

OutputCharset Property

Description

This property specifies the output character set for the Visual Designer. Both the data model and the views output by the Visual Designer will be encoded by the Visual Designer using the provided output character set. If no output character set is specified the `UTF-8` character set will be used.

Syntax

JavaScript

```
setOutputCharset(strCharset);
```

Parameters

strCharset	A string specifying the character set used by the Visual Designer when it outputs the data model and views. 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 Web site [http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html].

Examples

Example 15.24. OutputCharset Property Example Scripting

The following example specifies that the XSD and XSLs should be output using the ASCII character set.

JavaScript

```
designer.setOutputCharset("ASCII");
```

Remarks

If no output character set is specified the `UTF-8` character set will be used.

Preload Property

Description

This property can be used when preloading EditLive! for XML. Once EditLive! for XML has finished loading the JavaScript function defined by the **Preload** property is used as a callback.

Syntax

JavaScript

```
  setPreload(strCallBack);
```

Parameters

strCallBack	A string specifying the name of a JavaScript function to use as the callback function once EditLive! for XML has finished loading.

Example

Example 15.25. Preload JavaScript Callback Function

The following code provides the JavaScript callback function which will display an **alert** dialog once EditLive! for XML has finished loading. The callback function is named `preloadReturn`.

```
<script language="javascript">
  function preloadReturn(){
    alert("EditLive! has finished preloading.");
  }
```

```
</script>
```

Example 15.26. Preload Property Example Scripting

The following example instantiate a version of EditLive! for XML and assign 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.

JavaScript

```
var editlivejs;
editlivejs = new EditLiveXML("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! for XML. This can improve the performance of EditLive! for XML within a Web application. Preloading EditLive! for XML causes the browser's XML Plug-In and the EditLive! for XML classes to be loaded.

It is recommended that, when preloading EditLive! for XML, you set the height and width of the EditLive! for XML applet so they are both one pixel. This will ensure that the EditLive! for XML applet is not visible on the page.

Preloading EditLive! for XML can be performed on any page within a Web application.

ShowButtonIconURL Property

Description

This property specifies the URL at which the icon to be used with the *show* button for EditLive! for XML. This function can only be used when displaying EditLive! for XML as a button in the browser using the **ShowAsButton** method.

Syntax

JavaScript

```
setShowButtonIconURL(strImageURL);
```

Parameters

strImageURL	A string which is the URL for where the icon image for the *show* button for this instance of EditLive! for XML can be requested from.

Examples

Example 15.27. ShowButtonIconURL Property Example Scripting

The following code would specify that EditLive! for XML is to use the image `http://server.com/icons/showButton.gif` when displaying the *show* button for EditLive! for XML.

JavaScript

```
editlivejs.setShowButtonIconURL("http://server.com/icons/showButton.gif");
...
editlivejs.showAsButton("true");
```

Remarks

This function should only be used when the **ShowAsButton** method is used to display EditLive! for XML.

See Also

- ShowAsButton method
- ShowButtonText property
- HideButtonIconURL property

- HideButtonText property

ShowButtonText Property

Description

This property specifies the text to be used with the *show* button for EditLive! for XML. This function can only be used when displaying EditLive! for XML as a button in the browser using the **ShowAsButton** method.

Syntax

JavaScript

```
setShowButtonText(strFileURL);
```

Parameters

strButtonText	A string which is the text to be displayed on the *show* button for this instance of EditLive! for XML when it is displayed as a button.

Examples

Example 15.28. ShowButtonIconURL Property Example Scripting

The following code would specify that EditLive! for XML is to use the text `Show EditLive! for XML` when displaying the *show* button for EditLive! for XML.

JavaScript

```
editlivejs.setShowButtonText(escape("Show EditLive! for XML"));
...
editlivejs.showAsButton("true");
```

Note

The string passed to the JavaScript **setShowButtonText** property must be URL encoded. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

Remarks

This function should only be used when the **ShowAsButton** method is used to display EditLive! for XML.

The string passed to the JavaScript **setShowButtonText** property must be URL encoded. It is recommended that a server-side URL encoding function be used if available as the JavaScript **escape** function does not fully comply with the URL encoding standard.

See Also

- ShowAsButton method
- ShowButtonIconURL property
- HideButtonIconURL property
- HideButtonText property

ShowSystemRequirementsError Property

Description

This property specifies the way in which Ephox EditLive! for XML reacts when a client machine does not match the system requirements needed to run EditLive! for XML.

Syntax

JavaScript

```
setShowSystemRequirementsError(blnShowError);
```

Parameters

blnShowError	A boolean indicating if an error message should be displayed within a page if EditLive! for XML is not able to load due to the client system not meeting the EditLive! for XML system requirements. When set to `true` an error message will be displayed. The default value is `true`.

Examples

Example 15.29. ShowSystemRequirementsError Property Example Scripting

The following code will cause **no** error message to be displayed when a client machine does not meet the system requirements needed to run EditLive! for XML.

JavaScript

```
var editlivejs;
editlivejs = new EditLiveXML("ELApplet1","700","400");
editlivejs.setShowSystemRequirementsError(false);
```

Remarks

When a client machine does not meet the system requirements needed to run EditLive! for XML a text area will be displayed where the instance of EditLive! for XML would normally be. In addition to this, by default, an error message will also be displayed indicating that the client machine does not meet the system requirements for EditLive! for XML. This error message disabled by setting the **ShowRequirementsError** property to false.

The displaying of a text area in place of EditLive! for XML on a client machine which does not meet the system requirements of EditLive! for XML cannot be disabled.

UseWebDAV Property

Description

This property specifies whether the EditLive! for XML should make its WebDAV functionality available to users for use with images and hyperlinks.

Syntax

JavaScript

```
setUseWebDAV(blnWebDAV);
```

Parameters

blnWebDAV	A boolean indicating if EditLive! for XML should make the WebDAV functionality available to users for image and hyperlink insertion. The default value is `false`.

Examples

Example 15.30. UseWebDAV Property Example Scripting

The following code would inform EditLive! for XML to make the WebDAV functionality available to users.

JavaScript

```
var editlivejs;
editlivejs = new EditLiveXML("ELApplet1","700","400");
editlivejs.setUseWebDAV(true);
```

Remarks

When making EditLive! for XML's WebDAV functionality available to users EditLive! for XML must also be configured correctly via the EditLive! for XML configuration. The <repository> XML element contains the relevant configuration information for using a WebDAV repository with EditLive! for XML.

See Also

- <repository> Configuration Element

XSDAsText Property

Description

This property specifies an XML Schema Document to be used with an instance of EditLive! for XML. This property accepts an XSD as a text string. Loading an XSD as text is the only way an XSD can be loaded into the Visual Designer. This can be most easily achieved by using server-side scripting to load the XSD file from the server's file system into a scripting string variable which can then be used when instantiating the Visual Designer.

Note

This property differs from the **addXSDAsText** method in that it can only be used with the Visual Designer and can only be called once.

Syntax

JavaScript

```
setXSDAsText(strXSD);
```

Parameters

strXSD	A string which contains a full XSD to be used with this instance of the Visual Designer.

Note

The XSD string should be URL encoded. This can be achieved with a server side URL encoding method.

Examples

Example 15.31. XSDAsText Property Scripting Example

The following example adds an XSD for use with the Visual Designer.

JavaScript

```
designerjs.setXSDAsText = "escape(<xsd:schema targetNamespace=...");
```

Note

The XSD in the example above is incomplete, and will not function. It is given only as a example to aid understanding. The XSD string passed to the Visual Designer via this property should not contain carriage return or new line characters.

Remarks

The string parameter for this method must be URL encoded. It is recommended that a server-side URL encoding function be used if available. The usage of the JavaScript **escape** function is *not* recommended as the JavaScript **escape** function does not fully comply with the URL encoding standard.

XSDURL Property

Description

This property specifies the XML Schema Document(s) to be used with an instance of EditLive! for XML. The schema document(s) specify the data structure and data restrictions for the form.

Note

This property cannot be used with the Visual Designer. To set the XSD for use with the Visual Designer the **XSDAsText** property must be used.

Syntax

JavaScript

```
setXSDURL(strXSDURL);
```

Parameters

strXSDURL	A string specifying a list of XML Schema Documents (XSDs) to be used within this instance of EditLive! for XML. When specifying multiple XSDs they should be separated by a single whitespace character.

Examples

Example 15.32. XSDURL Property Example Scripting

The following example specifies three XSDs for use with an instance of EditLive! for XML.

JavaScript

```
editlive_js.setXSDURL = "../XSDs/xsdOne.xsd
           ../XSDs/xsdTwo.xsd http://server/XSDs/xsdThree.xsd";
```

Remarks

The URL(s) specified may be relative or absolute.

When using relative URLs the URLs will be considered as being relative to the URL of the page in which EditLive! for XML is embedded.

Chapter 16. JavaScript Runtime API

This section of the API guide includes functions which may be called during runtime for the EditLive! for XML or Visual Designer applets.

GetCharCount Function

Description

This function obtains a count of the number of characters present within the EditLive! for XML applet. It counts the number of characters which are contained within controls on the current view. This function takes the name of a JavaScript function as its only parameter.

Syntax

JavaScript

```
GetCharCount(strJSFunct);
```

Parameters

strJSFunct	The name of the function which receives the character count obtained from the EditLive! for XML applet.

Example

Example 16.1. GetCharCount Runtime Function Example

The following code first creates a JavaScript function which is to be used as the parameter for the **GetCharCount** function. The JavaScript function will display the character count of the EditLive! for XML applet in a JavaScript **alert** dialog. The **GetCharCount** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<html>
  <head>
    <title>EditLive! for XML JavaScript Example</title>
    <!--Include the EditLive! for XML JavaScript Library-->
    <script src="editlivexml/editlivexml.js" language="JavaScript">
    </script>
    <script language="JavaScript">
      function charCountAlert(src){
        alert('Character Count: '+src);
      }
    </script>
  </head>
  <body>
    <form name="exampleForm">
      <h1>Instance of EditLive! for XML</h1>
      <script language="JavaScript">
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",600 , 400);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(escape('<p>There are 51 characters in
this instance of EditLive! for XML</p>'));
        editlivejs.show();
      </script>
      <br/>
      <p>Click this button to obtain a character count</p>
      <input type="button" value="Character Count"
        onClick="editlivejs.GetCharCount('charCountAlert');"/>
    </form>
  </body>
</html>
```

GetDocument Function

Description

This function retrieves the entire contents of the EditLive! for XML applet. This function takes the name of a JavaScript function as its only parameter.

Syntax

JavaScript

```
GetDocument(strJSFunct,[blnUploadImages]);
```

Parameters

strJSFunct	This is a required parameter. The name of the JavaScript function which receives the retrieved EditLive! for XML applet contents.
blnUploadImages	This is an optional parameter. 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`.

Example

Example 16.2. GetDocument Runtime Function Example

The following code first creates a JavaScript function which is to be used as the parameter for the **GetDocument** function. The JavaScript function will save the retrieved contents to a `<TEXTAREA>` with the name `documentContents`. The **GetDocument** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`. Images in EditLive! for XML will be uploaded to the server when **GetDocument** is called.

```
<html>
  <head>
    <title>EditLive! for XML JavaScript Example</title>
    <!--Include the EditLive! for XML JavaScript Library-->
    <script src="editlivexml/editlivexml.js" language="JavaScript">
    </script>
    <script language="JavaScript">
      function retrieveDocument(src){
        document.exampleForm.documentContents.value = src;
      }
    </script>
  </head>
  <body>
    <form name="exampleForm">
      <p>EditLive! for XML contents will appear here</p>
      <!--Create a textarea to save the applet contents to-->
      <p><textarea name="documentContents" cols="40" rows="10">
```

```
</textarea></p>
      <p>Click this button to retrieve applet contents</p>
      <p><input type="button" name="button1" value="Retrieve Contents"
onClick="editlivejs.GetDocument('retrieveDocument',true);"/></p>
      <!--Create an instance of EditLive! for XML-->
      <script language="JavaScript">
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",450 , 275);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(escape('<html><body><p>Some initial
text</p></body></html>'));
        editlivejs.show();
      </script>
    </form>
  </body>
</html>
```

Remarks

When uploading locally stored images to the relevant Web server for an instance of EditLive! for XML ensure that the *`blnUploadImages`* parameter is set to `true` when calling the **GetDocument** function.

GetSelectedText Function

Description

This function retrieves the currently selected text within the EditLive! for XML applet. The function 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. This function takes the name of a JavaScript function as its only parameter.

Syntax

JavaScript

```
  GetSelectedText(strJSFunct);
```

Parameters

strJSFunct	This is a required parameter. The name of the JavaScript function which receives the retrieved selected text.

Example

Example 16.3. GetSelectedText Runtime Function Example

The following code first creates a JavaScript function (`retrieveSelectedText`) which is to be used as the parameter for the **GetSelectedText** function. The JavaScript function will save the retrieved text to a `<TEXTAREA>` with the name `selectedText`. The **GetSelectedText** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<html>
  <head>
    <title>EditLive! for XML JavaScript Example</title>
    <!--Include the EditLive! for XML JavaScript Library-->
    <script src="editlivexml/editlivexml.js" language="JavaScript">
    </script>
    <script language="JavaScript">
      function retrieveSelectedText(src){
        document.exampleForm.selectedText.value = src;
      }
    </script>
  </head>
  <body>
    <form name="exampleForm">
      <p>EditLive! for XML contents will appear here</p>
      <!--Create a textarea to save the selected text to-->
      <p><textarea name="selectedText" cols="40" rows="10">
</textarea></p>
      <p>Click this button to retrieve applet contents</p>
      <p><input type="button" name="button1" value="Retrieve Contents"
onClick="editlivejs.GetSelectedText('retrieveSelectedText');"/></p>
      <!--Create an instance of EditLive! for XML-->
      <script language="JavaScript">
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",450 , 275);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(escape('<html><body><p>Some initial
text</p></body></html>'));
        editlivejs.show();
      </script>
    </form>
  </body>
</html>
```

Remarks

This function is designed for use when selecting text within a single parent block tag. Using the function outside of this context may result in unexpected behavior.

If using this function 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! for XML:

```
<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
```

Note

The **GetSelectedText** function is designed for use when selecting text within a single parent block tag. Using the function outside of this context may result in unexpected behavior and is not recommended.

GetWordCount Function

Description

This function obtains a count of the number of words present within the EditLive! for XML applet. It counts the number of words which are contained within controls on the current view. This function takes the name of a JavaScript function as its only parameter.

Syntax

JavaScript

```
GetWordCount(strJSFunct);
```

Parameters

strJSFunct	The name of the function which receives the word count obtained from the EditLive! for XML applet.

Example

Example 16.4. GetWordCount Runtime Function Example

The following code first creates a JavaScript function which is to be used as the parameter for the **GetWordCount** function. The JavaScript function will display the word count of the EditLive! for XML applet in a JavaScript **alert** dialog. The **GetWordCount** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<html>
  <head>
    <title>EditLive! for XML JavaScript Example</title>
    <!--Include the EditLive! for XML JavaScript Library-->
    <script src="editlivexml/editlivexml.js" language="JavaScript">
    </script>
    <script language="JavaScript">
      function wordCountAlert(src){
        alert('Word Count: '+src);
      }
    </script>
  </head>
  <body>
    <form name="exampleForm">
      <!--Create an instance of EditLive! for XML-->
      <h1>Instance of EditLive! for XML</h1>
      <script language="JavaScript">
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",600 , 400);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(escape('<p>There are 11 words in this
instance of EditLive! for XML</p>'));
        editlivejs.show();
      </script>
      <br/>
      <p>Click this button to obtain a word count</p>
      <input type="button" value="Word Count"
        onClick="editlivejs.GetWordCount('wordCountAlert');"/>
    </form>
  </body>
</html>
```

InsertHTMLAtCursor Function

Description

This function inserts developer specified HTML at the cursor within the EditLive! for XML applet. This function takes a JavaScript string as its only parameter.

Note

This function will only work whilst the cursor is placed within an XHTML section within EditLive! for XML.

Syntax

JavaScript

```
  InsertHTMLAtCursor(strHTML);
```

Parameters

strHTML The string containing the HTML to be inserted at the cursor within the EditLive! for XML applet.

Note

This string must be URL encoded. It is recommended that this encoding is done via a server-side URL encoding method. It can be achieved using the JavaScript **escape** function, however, this is not recommended as the **escape** function does.

Example

Example 16.5. InsertHTMLAtCursor Runtime Function Example

The following code creates a `<TEXTAREA>`, named `htmlToInsert`, that will have its contents inserted into an instance of EditLive! for XML at the cursor position via the **InsertHTMLAtCursor** function. The **InsertHTMLAtCursor** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.js" language="JavaScript">
    </SCRIPT>
    </HEAD>
    <BODY>
      <FORM name = exampleForm>
        <P>EditLive! for XML 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(escape(
document.exampleForm.htmlToInsert.value
));">
        </P>
        <!--Create an instance of EditLive! for XML-->
        <SCRIPT language="JavaScript">
          <!--
          var editlivejs;
          editlivejs = new EditLiveXML("editlive",450 , 275);
          editlivejs.setDownloadDirectory("editlivexml");
          editlivejs.setLocalDeployment(false);
          editlivejs.setConfigurationFile("sample_elconfig.xml");
          editlivejs.setDocument(escape("<P>This is EditLive!</P>"));
          editlivejs.show();
        -->
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

InsertHyperlinkAtCursor Function

Description

This function applies a developer specified hyperlink to the text selected within EditLive! for XML. 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! for XML). The first argument of this function must be the URL corresponding to the hyperlink, this argument is mandatory. This function 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.

Note

In EditLive! for XML the **InsertHyperlinkAtCursor** function applies when the cursor is within an XHTML section of the XML document.

Syntax

JavaScript

```
  InsertHyperlinkAtCursor(strHyperlink, [strAttribute]*);
```

Parameters

strHyperlink	This is a required parameter. This string contains the hyperlink to be inserted at the cursor within the EditLive! for XML 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

Example 16.6. InsertHyperlinkAtCursor Runtime Function Examples

The following code creates a text `<INPUT>`, named *`hyperlinkToInsert`*, that will have its contents inserted into an instance of EditLive! for XML at the cursor position via the **InsertHyperlinkAtCursor** function. The **InsertHyperlinkAtCursor** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.js" language="JavaScript">
     </SCRIPT>
    </HEAD>
    <BODY>
      <FORM name = exampleForm>
      <P>The selected hyperlink will be inserted into EditLive! for XML</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! for XML
          <OPTION value="mailto:someone@yourserver.com">Email Link
        </SELECT>
      </P>
      <P>Click here to insert hyperlink at the cursor in EditLive! for XML</P>
      <P>
        <INPUT type="button" name="button1" value="Insert Hyperlink"
onClick="editlive1.InsertHyperlinkAtCursor(document.exampleForm.
hyperlinkToInsert.value);">
      </P>
      <!--Create an instance of EditLive! for XML-->
      <SCRIPT language="JavaScript">
        <!--
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",450 , 275);
        editlivejs.setDownloadDirectory("editlive");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_config.xml");
        editlivejs.setBody(escape("<p>Contents of EditLive! for XML</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! for XML. 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 function can be used with differing numbers of arguments. However, the first argument of the function 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 usage of the correct quotations marks.

IsValid Function

Description

This function checks the validity of the content contained in the EditLive! for XML applet.

Syntax

JavaScript

```
IsValid(strJSFunct);
```

Parameters

strJSFunct	This is a required parameter. The name of the JavaScript callback function which receives the boolean result of the **IsValid** call.

Example

Example 16.7. IsValid Runtime Function Example

The following example demonstrates how to use the **IsValid** method in association with an instance of EditLive! for XML. In the example a button that calls the **IsValid** function is used to check the validity of the document in EditLive! for XML. The result of this check is presented to the user in a text field.

```
<script language="JavaScript">
  function receiveIsValid(isValidBln){
    document.exampleForm.isValidResult.value = isValidBln;
  }
</script>

...

<form name="exampleForm">
  <p>EditLive! for XML validity check result will appear here:</p>
  <!--Create a text field to send the result of the IsValid call to-->
  <p><input type="text" name="isValidResult" width="40"></p>

  <p>Click this button check the validity of the current content</p>
  <p><input type="button" name="button1" value="Check Validity"
onClick="editlivejs.IsValid('receiveIsValid');"/></p>

  <!--Create an instance of EditLive! for XML-->
  <script language="JavaScript">
    var editlivejs;
    editlivejs = new EditLiveXML("editlive",700 , 600);
    ...
    editlivejs.show();
  </script>
</form>
```

PostDocument Function

Description

This function allows developers to cause EditLive! for XML to use HTTP POST to submit its content directly to a POST acceptor script. EditLive! for XML will also process the HTTP response. The response is processed in accordance with a parameter passed to the JavaScript method.

Syntax

JavaScript

```
  PostDocument(strFieldName, strPostURL, strResponseProcessing,
              [strJSFunctionName]);
```

Parameters

strFieldName	This parameter is required. The name of the field in the HTTP POST that EditLive! for XML uses to POST its content.
strPostURL	This parameter is required.

	The URL for the POST acceptor script that EditLive! for XML POSTs to.
strResponseProcessing	This parameter is required. This parameter indicates how EditLive! for XML should process the response. It has the following possible values: • `saveToDisk` • `callback` **Note** When setting this parameter to `callback` the *`strJSFunctionName`* must also be specified.
strJSFunctionName	This parameter is optional. The name of the JavaScript function to be used as a callback function. The JavaScript function specified should accept the content of the HTTP response as its only parameter. This parameter should be set in conjunction with setting the *`strResponseProcessing`* parameter to `callback`.

Example

Example 16.8. POSTDocument Runtime Function Example

The following code creates a button called `Save` on a HTML page. When the button is clicked it causes the instance of EditLive! for XML to POST its content to `http://someserver/post/postacceptor.jsp` in the field named `editliveField`. Upon receiving the HTTP response EditLive! for XML presents the user with a dialog allowing them to save the content of the response to disk. This is performed by setting the *`strResponseProcessing`* parameter to `saveToDisk`

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.js" language="JavaScript">
    </SCRIPT>
    </HEAD>
    <BODY>
      <FORM name = exampleForm>
        <P>Click this button to POST the document in EditLive!</P>
        <P>
          <INPUT type="button" name="button1" value="Save"
onClick="editlivejs.PostDocument('editliveField',
'http://someserver/post/postacceptor.jsp', 'saveToDisk');">
        </P>
        <!--Create an instance of EditLive! for XML-->
        <SCRIPT language="JavaScript">
          <!--
          var editlivejs;
          editlivejs = new EditLiveXML("editlive",450 , 275);
          editlivejs.setDownloadDirectory("editlivexml");
          editlivejs.setLocalDeployment(false);
          editlivejs.setConfigurationFile("sample_elconfig.xml");
          editlivejs.setDocument(escape("<P>This is EditLive!</P>"));
          editlivejs.show();
        -->
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

SetDocument Function

Description

This function sets the contents of the EditLive! for XML applet. It will replace any existing contents of the applet with the contents the function is provided with as its parameter. This function takes a JavaScript string as its only parameter.

Syntax

JavaScript

```
SetDocument(strDocument);
```

Parameters

strDocument	A string of the contents to be placed into the EditLive! for XML applet.

Note

This string must be URL encoded. It is recommended that this encoding is done via a server-side URL encoding method. It can be achieved using the JavaScript **escape** function, however, this is not recommended as the **escape** function does.

Example

Example 16.9. SetDocument Runtime Function Example

The following code creates a `<TEXTAREA>`, named `documentContents`, that will have its contents loaded into an instance of EditLive! for XML via the **SetDocument** function. The **SetDocument** function will be associated with a HTML button. The name of the EditLive! for XML applet is `editlivejs`.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.js" language="JavaScript"></SCRIPT>
  </HEAD>
  <BODY>
    <FORM name = exampleForm>
      <P>EditLive! for XML 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>Click this button to set applet contents</P>
        <INPUT
          type="button" name="button1" value="Set Contents"
onClick="editlivejs.SetDocument(escape(
document.exampleForm.documentContents.value));">
      </P>
      <!--Create an instance of EditLive! for XML-->
      <SCRIPT language="JavaScript">
        <!--
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",450 , 275);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_config.xml");
        editlivejs.show();
        -->
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

SetProperties Function

Description

This function is for use in conjunction with the custom properties dialog. This function, when given a string of relevant name-value pairs, sets the attributes for an instance of a specific tag within EditLive! for XML. For more information on how to use custom properties dialogs with EditLive! for XML please see Custom Properties Dialogs for EditLive! for XML.

Syntax

JavaScript

```
  SetProperties(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 a n`equals (=)` character. Name-value pairings must be delimited by a new line (`\n`) character.

Note

This string must be URL encoded. It is recommended that this encoding is done via a server-side URL encoding method. Encoding can be achieved using the JavaScript **escape** function, however, this is not recommended as the **escape** function does not fully comply with the URL encoding standards.

Example

Example 16.10. SetProperties Runtime Function Example

The following sets the properties for an instance of a tag inside an instance of EditLive! for XML named `editlivejs`.

```
//set up an instance of EditLive!
var editlivejs;
editlivejs = new EditLiveXML("editlive",700,400);
editlivejs.setDownloadDirectory("editlivexml");
editlivejs.setLocalDeployment(false);
editlivejs.setConfigurationFile("sample_config.xml");
editlivejs.show();
...
//create a function which sets properties
function setNewProperties(newProperties){
  editlivejs.SetProperties(newProperties);
}
```

Remarks

Each name and value for each pairing must be delimited by an equals (`=)` 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! for XML. 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! for XML. 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 for which the custom properties dialog applies 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

- Custom Properties Dialogs for EditLive! for XML

SetXMLNodeValue Function

Description

This function is for use in conjunction with an `ephox:button` element embedded in a form for EditLive! for XML that uses the `getCurrentNodeValue` action. This function will allow the developer to set the value of an XML node given the `ID` of that node as supplied by the `ephox:button` using the `getCurrentNodeValue` action.

Syntax

JavaScript

```
SetProperties(intID, strValue);
```

Parameters

intID	An integer specifying the ID of the relevant XML node.
strValue	A string containing the value to set for the XML node.

Note

This string must be URL encoded. It is recommended that this encoding is done via a server-side URL encoding method. Encoding can be achieved using the JavaScript **escape** function, however, this is not recommended as the **escape** function does not fully comply with the URL encoding standards.

Example

Example 16.11. SetProperties Runtime Function Example

The following sets the properties for an instance of a tag inside an instance of EditLive! for XML named `editlivejs`.

```
//set up an instance of EditLive!
var editlivejs;
editlivejs = new EditLiveXML("editlive",700,400);
editlivejs.setDownloadDirectory("editlivexml");
editlivejs.setLocalDeployment(false);
editlivejs.setConfigurationFile("sample_config.xml");
...
editlivejs.show();
...
//create a function which sets properties
function setNewNodeValue(nodeID,newValue){
  editlivejs.SetXMLNodeValue(nodeID, newValue);
}
```

Remarks

In order to correctly set the properties of the relevant node it should be ensured that the **`id`** value supplied by the `getCurrentNodeValue` action is not altered by the functions external to EditLive! for XML.

UploadImages Function

Description

This function uploads any local images, within the instance of EditLive! for XML, to the Web server. Upon calling this function 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.

Syntax

JavaScript

```
UploadImages();
```

Parameters

This function takes no parameters.

Example

Example 16.12. UploadImages Runtime Function Example

The following code allows the user to force an upload of the images in EditLive! for XML to the Web server. The **UploadImages** function is called by clicking on a button within the form.

```
<HTML>
  <HEAD>
    <TITLE>EditLive! for XML JavaScript Example</TITLE>
    <!--Include the EditLive! for XML JavaScript Library-->
    <SCRIPT src="editlivexml/editlivexml.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! for XML-->
      <SCRIPT language="JavaScript">
        <!--
        var editlivejs;
        editlivejs = new EditLiveXML("editlive",450 , 275);
        editlivejs.setDownloadDirectory("editlivexml");
        editlivejs.setLocalDeployment(false);
        editlivejs.setConfigurationFile("sample_elconfig.xml");
        editlivejs.setDocument(escape("<p>Some initial text</p>"));
        editlivejs.show();
        -->
      </SCRIPT>
    </FORM>
  </BODY>
</HTML>
```

Remarks

The **UploadImages** function need not be called if using the EditLive! for XML **onSubmit** functionality.

The upload script used by this function is that specified in the EditLive! for XML configuration information in the **<httpUploadData>** element.

See Also

- **<httpUploadData>** XML element
- **GetDocument** JavaScript Function

Chapter 17. XML Configuration API

This chapter provides a reference for the EditLive! for XML XML Configuration API. The chapter lists all the XML elements which can be used within the EditLive! for XML in the order that they can appear in the configuration file.

<editLive> Configuration Element

This is the root element to be used in every EditLive! for XML 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. **Note** The configuration information placed within this element applies to the generated document layout or view only.
<ephoxLicenses>	The ephoxLicenses element handles the configuration of licensing information. Licensing information is used to ensure that you are using EditLive! for XML correctly. Ephox will provide you with the appropriate license files, these simply have to be added here.
<spellCheck>	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 **Code View** of EditLive! for XML.
<wysiwygEditor>	This element allows you to set options relating to the EditLive! for XML 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 in addition to configuring how EditLive! behaves with respect to images, both on the server and local machines.
<authentication>	The authentication structure allows for the configuration of authentication information for use with EditLive! for XML. 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! for XML.
<menubar>	The menuBar structure allows for the configuration of the EditLive! for XML menus.

Note

This structure configures only the menu bar and not the shortcut menu.

<toolbars>	The toolbars structure allows for the configuration of the EditLive! for XML format and command toolbars.
<shortcutMenu>	The shortcutMenu element allows for the customization of the

EditLive! for XML shortcut menu.

Remarks

Each EditLive! for XML configuration file must contain exactly one **`<editLive>`** element.

<document> Configuration Element

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 Ephox EditLive! for XML. This includes any META information and style sheet links.

Note

When configuring the Visual Designer with these settings the information contain within these settings becomes part of the views which are output by the Visual Designer.

Note

The settings which are configured through the **`<document>`** element will only appear in the view within EditLive! for XML. In EditLive! for XML the properties set within this element are not part of the output.

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! for XML 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.

<html> Configuration Element

This element allows for the configuration of information which is contained between the HEAD tags and as attributes within the BODY tag of the Ephox EditLive! for XML document. This includes any META information and style sheet links.

The settings which are configured through the **`<document>`** element will appear in the actual document inside EditLive! for XML.

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! for XML 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.

<head> Configuration Element

This element allows for the configuration of information which is to be contained between the `<HEAD>` tags of the EditLive! for XML document.

The settings which are configured through the **`<head>`** element will appear in the actual document inside EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - `<document>`
    - `<html>`
      - **`<head>`**

```
<editLive>
    <document>
        <html>
            <head>
                <!--head configuration settings-->
            </head>
            ....
        </html>
    </document>
    ...
</editLive>
```

Child Elements

<link>	This element provides the style sheet information which is to be stored in a `<LINK>` tag within the `<HEAD>` tag of the EditLive! for XML document. **Note** 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 Ephox EditLive! for XML document. The element has no attributes.

Remarks

The **`<head>`** element can appear only once within the **`<html>`** element.

<base> Configuration Element

This element provides the information which is to be stored as attributes within the BASE tag between the HEAD tags of the Ephox EditLive! for XML document. The element has only a **HREF** attribute.

The value which appears within the **`<base>`** element will appear within the actual EditLive! for XML 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.

Example

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! for XML will use the base location of the page which it resides in as the base URL. For example, if the instance of EditLive! for XML appeared in the page `http://www.yourserver.com/editlive.html` then the instance of EditLive! for XML 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

- BaseURL Property

<link> Configuration Element

This element provides the information which is to be stored as attributes within the LINK tag between the `<HEAD>` tags of the Ephox EditLive! for XML document.

The value which appears within the **`<link>`** element will appear within the actual EditLive! for XML 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! for XML 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! for XML 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! for XML 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! for XML 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 EditLive! for XML and Style Sheets document.

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

- EditLive! for XML and Style Sheets

<style> Configuration Element

This element provides the information which is to be stored between the `<STYLE>` tags, between the `<HEAD>` tags of the Ephox EditLive! for XML 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! for XML. 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! for XML are resolved according to the CSS precedence rules. Thus, 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! for XML 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.

<ephoxLicenses> Configuration Element

This structure contains the various licenses for Ephox EditLive! for XML. If this structure is left blank or if the licensing information contained within this element is incorrect then EditLive! for XML 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! for XML.

Example

This example demonstrates how to use an empty **<ephoxLicenses>** element to run EditLive! for XML 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! for XML will only run in the 30 day trial mode.

<license> Configuration Element

This element contains the configuration information for a single license of Ephox EditLive! for XML.

Note

Attributes within this element should be entered as per the license file provided by Ephox. If the configuration information provided by the attributes does not correspond to a valid license provided by Ephox then EditLive! for XML 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! for XML is licensed.
expiration	This attribute provides the expiration date of the license.
key	This attribute provides the product key for this license of EditLive! for XML.
licensee	This attribute provides the company or organisation to which this copy of EditLive! for XML is licensed.

product	This attribute details the Ephox product which can be used with this license.
release	This attribute details the release number of the Ephox 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 Ephox account ID.
activationURL	This attribute configures the URL that EditLive! for XML should use to check its license. If left blank the default value will be used. *Default Value:* `http://www.ephox.com/activate/eljf10.asp`
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. *Default Value:* `true` **Note** This attribute is a boolean and can only be `true` or `false`.
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. *Default Value:* `production` **Note** This attribute has the possible values of `production`, `development` or `QA`.

Example

This example demonstrates how to use the **`<license>`** element to input licensing information into EditLive! for XML.

```
<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 XML" 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=... />
```

<spellCheck> Configuration Element

This element defines the location of the JAR file to be used with the spell checker. This enables the dictionary to be defined for Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - **`<spellCheck>`**

```
<editLive>
     ...
     <spellCheck ... />
     ...
```

```
</editLive>
```

Optional Attributes

jar The value for this attribute corresponds to the location of the JAR file to be used with EditLive! for XML for spell checking.

> **Note**
>
> For the spell checking in EditLive! for XML to function a JAR file must be specified. The name of the JAR file for the spell checker dictionary must be in all lower case letters.

Example

The following example demonstrates how to define the location of the spell checker JAR file to be used with EditLive! for XML.

```
<editLive>
    ...
    <spellCheck
       jar="../../redistributables/editlivexml/dictionaries/en_us_3_1.jar"
     />
    ...
</editLive>
```

Remarks

A JAR file must be specified for the spell checker in order for the spell checking functionality of EditLive! for XML to work.

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=... />
```

<htmlFilter> Configuration Element

This element provides the HTML filter settings for use within Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - **`<htmlFilter>`**

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

Required Attributes

wrapLength	Specifies the number of characters for each line within the HTML source.

Note

Setting the **wrapLength** to zero (0) turns wrapping off within the HTML source.

Optional Attributes

indentContent	This option specifies that EditLive! for XML 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`

Note

	This attribute is a boolean and can only be `true` or `false`.
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` **Note** This attribute is a boolean and can only be `true` or `false`.
removeFontTags	If set to `true` EditLive! for XML will discard all font tags. If left blank the default value will be used. *Default Value:* `true` **Note** This attribute is a boolean and can only be `true` or `false`.
quoteMarks	If set to `true` quotation marks (") will be escaped and therefore appear as `&quot;`. If left blank the default value will be used. *Default Value:* `true` **Note** This attribute is a boolean and can only be `true` or `false`.
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 XHTML sections is valid. *Default Value:* `true`

Note

This attribute is a boolean and can only be `true` or `false`.

Note

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 XHTML sections. This will preserve the unknown tags. When set to `false` unknown tags will be HTML encoded i.g. "`<unknownTag>`" would be converted to "`&lt;UnknownTag&gt;`"

Default Value: `true`

Note

This attribute is a boolean and can only be `true` or `false`.

Note

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! for XML is to be run through a validation process then it is recommended that this attribute be set to `false`.

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! for XML 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 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:

```
<htmlFilter wrapLength=... />
```

<wysiwygEditor> Configuration Element

This element allows you to set options relating to the EditLive! for XML 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. *Default Value:* `top` Possible Values: • `top` - Places the tabs at the top of the editor pane. • `bottom` - Places the tabs at the bottom of the editor pane. • `off` - Removes the tabs from the editor.

Child Elements

<xmlEditor>	This structure allows for the configuration of the user interface components of EditLive! for XML.

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=.../>
```

<xmlEditor> Configuration Element

This element allows the configuration of the user interface for the EditLive! for XML editor.

Configuration Element Tree Structure

- `<editLive>`
  - `<wysiwygEditor>`
    - **`<xmlEditor>`**

```
<editLive>
  ...
  <wysiwygEditor>
    <xmlEditor ... />
  </wysiwygEditor>
  ...
</editLive>
```

Optional Attributes

showDocumentNavigator	Configures whether the XML document navigation bar is displayed immediately below the EditLive! for XML toolbars. If set to `true` then the document navigation bar will be displayed. *Default value:* `true`
showValidationPane	Configures whether the XML validation pane is displayed. The validation pane provides users with validation information during runtime. There are three possible values for this attribute: • `auto` - When set to `auto` the validation pane will be displayed to users only when there is a validation error.

- `true` - When set to `true` the validation pane will be displayed to users all the time.
- `false` - When set to `false` the validation pane will never be displayed to users.

Example

The following example would hide the validation pane from users at all times, it would also cause the document navigator to be hidden.

```
<editLive>
  ...
  <wysiwygEditor tabPlacement="bottom">
    <xmlEditor
      showValidationPane="false"
      showDocumentNavigator="false"
    />
  </wysiwygEditor>
  ...
</editLive>
```

Remarks

The **`<xmlEditor>`** element can appear only once within the **`<wysiwygEditor>`** element.

If the **`<xmlEditor>`** 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:

```
<xmlEditor showValidationPane=.../>
```

<sourceEditor> Configuration Element

This element provides the code view settings for use within Ephox EditLive! for XML.

Note

This element does not apply for the configuration of the Visual Designer.

Configuration Element Tree Structure

- `<editLive>`
  - **`<sourceEditor>`**

```
<editLive>
     ...
     <sourceEditor ... />
     ...
</editLive>
```

Optional Attributes

enabled	If set to `true` users will be able to access the **Code** view for documents.

Example

The following example would allow users to view the source code of a document within EditLive! for XML.

```
<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"/>
```

<wordImport> Configuration Element

This element configures the manner in which Ephox EditLive! for XML 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! for XML upon detecting an import from Microsoft Word. This attribute has three possible values; `user_prompt`, `merge` or `clean`. • The `user_prompt` setting will result in EditLive! for XML prompting the user as to whether they wish to import Microsoft Word styles or strip them from the imported text. • The `merge` setting will result in the Microsoft Word styles being merged with the current style information within EditLive! for XML. • The `clean` setting will result in EditLive! for XML stripping the Microsoft Word Styles from the imported text.

Note

Styles imported from Microsoft Word *will not* overwrite styles which already exist within the document.

Note

For EditLive! for XML this should always be set to `clean`.

Example

The following example demonstrates how to set EditLive! for XML 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=.../>
```

<mediaSettings> Configuration Element

This element allows for the configuration of media settings, such as image settings, within Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`

  - **`<mediaSettings>`**

```
<editLive>
    <mediaSettings>
        <!--mediaSettings configuration settings-->
    </mediaSettings>
    ...
</editLive>
```

Child Elements

<images>	This element allows for the configuration of information for images and any other media within EditLive! for XML.

Remarks

The **`<mediaSettings>`** element can appear only once within the **`<editLive>`** element.

<images> Configuration Element

This element allows for the configuration of information for images and any other media within Ephox EditLive! for XML.

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. **Note** 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. **Note** If a user specifies the URL for a local file then EditLive! for XML will attempt to upload this file.

Child Elements

<httpImageUpload>	This element provides the configuration information to be used for HTTP image upload within EditLive! for XML.
<imageList>	This element provides a list of images stored on a Web server which can be accessed by the end users of EditLive! for XML.
<webdav>	This element allows for the customization of the EditLive! for

XML WebDAV functionality.

Remarks

The **`<images>`** element can appear only once within the **`<mediaSettings>`** element.

<httpImageUpload> Configuration Element

This element allows for the configuration of information which is used when using HTTP image upload within Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - `<mediaSettings>`
    - `<images>`
      - **`<httpImageUpload>`**

```
<editLive>
     ...
     <mediaSettings>
          <images>
               <httpImageUpload ... />
               ...
          </images>
     </mediaSettings>
     ...
</editLive>
```

Required Attributes

base	This attribute defines the location where images can be found *after* they have

	been uploaded. For more information please see the article on HTTP image upload.
href	This attribute defines the location on the Web server of the script which handles image uploads. For more information please see the article on HTTP image upload.

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. Information is provided in name and value pairings.

Example

The following example demonstrates how to define the **base** and **href** attributes for EditLive! for XML.

```
<editLive>
    ...
    <mediaSettings>
        <images>
            <httpImageUpload
                base="http://yourserver.com/imagedir/"
                href="http://yourserver.com/scripts/uploadhandler.asp"
             />
            ...
        </images>
    </mediaSettings>
    ...
</editLive>
```

Remarks

The **`<httpImageUpload>`** element can appear only once within the **`<images>`** element.

If there is no **`<httpUploadData>`** element(s) then **`<httpImageUpload>`** 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:

```
<httpImageUpload base=... />
```

If there are **`<httpUploadData>`** elements present then the **`<httpImageUpload>`** element needs to have both opening and closing tags. It should appear as below:

```
...
<httpImageUpload base=... >
     <httpUploadData name=... />
     <httpUploadData name=... />
</httpImageUpload>
...
```

See Also

- Using HTTP for Image Upload in EditLive! for XML

<httpUploadData> Configuration Element

This element allows for the configuration of information which is used when using HTTP image upload within Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - `<mediaSettings>`
    - `<images>`
      - `<httpImageUpload>`
        - **`<httpUploadData>`**

```
<editLive>
     ...
     <mediaSettings>
          <images>
               <httpImageUpload ... >
                    <httpUploadData ... />
               </httpImageUpload>
               ...
          </images>
     </mediaSettings>
     ...
</editLive>
```

Required Attributes

name	This attribute should contain the name for the extra data 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>
        <images>
            <httpImageUpload
               base="http://yourserver.com/imagedir/"
               href="http://yourserver.com/scripts/uploadhandler.asp"
            >
                 <httpUploadData name="user" data="default"/>
                 <httpUploadData name="priority" data="1"/>
            ...
            </httpImageUpload>
            ...
        <images>
    <mediaSettings>
    ...
</editLive>
```

Remarks

The **`<httpUploadData>`** element can appear multiple times within the **`<httpImageUpload>`** 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=... />
```

<imageList> Configuration Element

This element contains a list of all the server images to be configured for use with EditLive! for XML.

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! for XML.

Remarks

The **`<imageList>`** element can appear only once within the **`<images>`** element.

<image> Configuration Element

This element defines the properties of an image stored on a server which is to be used with Ephox EditLive! for XML.

Configuration Element Tree Structure

- `<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 `<httpImageUpload>` 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! for XML 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! for XML 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! for XML 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! for XML 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! for XML.
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! for XML 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! for XML 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! for XML 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! for XML.
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! for XML 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! for XML document this attribute will appear in the source code.

Example

The following example demonstrates how to configure a server image for use with EditLive! for XML.

```
<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=... />
```

<webdav> Configuration Element

This element contains the configuration information for the use of Ephox EditLive! for XML with WebDAV repositories.

Configuration Element Tree Structure

- `<editLive>`

- `<mediaSettings>`
  - `<images>`
    - **`<webdav>`**

OR

- `<editLive>`
  - `<hyperlinks>`
    - **`<webdav>`**

```
<editLive>
     ...
     <mediaSettings>
          <images>
               ...
               <webdav
                    <!--webdav configuration settings-->
               </webdav>
          </images>
     </mediaSettings>
     ...
</editLive>

OR

<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.

To use the WebDAV functionality of EditLive! for XML the WebDAV property for EditLive! for XML in your API must be set. For more information please consult the reference of your EditLive! for XML API.

<repository> Configuration Element

This element defines the configuration settings for the use of Ephox EditLive! for XML with a WebDAV repository.

Configuration Element Tree Structure

- `<editLive>`
  - `<mediaSettings>`
    - `<images>`
      - `<webdav>`
        - **`<repository>`**

OR

- `<editLive>`
  - `<hyperlinks>`
    - `<webdav>`
      - **`<repository>`**

```
<editLive>
     ...
     <mediaSettings>
          <images>
               ...
               <webdav
                    <repository ... />
               </webdav>
          </images>
     </mediaSettings>
     ...
</editLive>

OR

<editLive>
     ...
     <hyperlinks>
          ...
          <webdav>
              <repository ... />
          </webdav>
          ...
     </hyperlinks>
     ...
</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! for XML concerned.

Note

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! for XML concerned.

Optional Attributes

name	The human-readable name for this WebDAV repository.
defaultDir	The initial directory that EditLive! for XML is to access on the WebDAV server.
useMimeType	Whether or not to filter files according to their mime type. If this is false, the files are filtered according to their file extension. *Default:* The default value for this attribute is `true`.

Note

This attribute has two possible values; `true` or `false`.

Example

The following example demonstrates how to define a WebDAV repository with the root URL `http://www.yourserver.com/webDAV` for use with an instance of EditLive! for XML 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 with the root URL `http://www.yourserver.com/webDAV` for use with an instance of EditLive! for XML running on a server different to that of the WebDAV repository. Thus, an absolute URL must be used.

```
<editLive>
    ...
    <mediaSettings>
        <images>
            <webdav>
                <repository name="Sample"
                    baseDir="http://www.yourserver.com/webDAV"
                    defaultDir="SampleDir"
                    webDAVBaseURL="http://www.yourserver.com/webDAV"
                />
            </webdav>
        </images>
    </mediaSettings>
    ...
</editLive>
```

Remarks

For WebDAV repositories requiring user authentication the **`<realm>`** element should be used to specify the user name and password for the repository.

The **`<repository>`** element can appear multiple times within the **`<webdav>`** element.

The **`<repository>`** 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:

```
<repository name=... />
```

The first repository listed in the EditLive! for XML configuration file is the default WebDAV repository.

See Also

- **<realm>** Element

<authentication> Configuration Element

This element contains the settings required for a user of EditLive! for XML to authenticate themselves to a specific Web server realm. The element should contain the settings for each realm that the instance of EditLive! for XML concerned may access.

Configuration Element Tree Structure

- `<editLive>`
  - **`<authentication>`**

```
<editLive>
    <authentication>
        <!--authentication configuration settings-->
    </authentication>
    ...
</editLive>
```

Child Configuration 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.

<realm> Configuration Element

This element contains the settings required for a user to authenticate themselves to a realm on a Web server. EditLive! for XML 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.

Note

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.

Note

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 XML configuration the end user will be prompted for the details required. The details provided in the XML, if any, will be supplied to the end user when prompted. For example, if only the username and domain are supplied by the XML configuration the end user will be prompted and will only have to supply the correct password.

<hyperlinks> Configuration Element

This element allows for the configuration of a list of hyperlinks and e-mail addresses to be made available to the end users of Ephox EditLive! for XML 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.
<mailtoList>	This element allows for the specification of the list of e-mail addresses that the end users will be provided with.
<webdav>	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.

<hyperlinkList> Configuration Element

This element allows for the specification of a list of hyperlinks that the users of Ephox EditLive! for XML 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>
```

Child Elements

<hyperlink>	This element defines a hyperlink that users of EditLive! for XML will be provided with via the **Insert Hyperlink** dialog.

Remarks

The **`<hyperlinkList>`** element can appear only once within the **`<hyperlinks>`** element.

<hyperlink> Configuration Element

This element allows for the specification of a single hyperlink that the end users of Ephox EditLive! for XML 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! for XML.
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! for XML 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! for XML 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! for XML 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=... />
```

<mailtoList> Configuration Element

This element allows for the specification of a list of e-mail addresses that the end users of Ephox EditLive! for XML 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>
```

Child Elements

<mailtolink>	This element defines an e-mail address that end users of EditLive! for XML will be provided with via the **Insert Hyperlink** dialog.

Remarks

The **`<mailtoList>`** element can appear only once within the **`<hyperlinks>`** element.

<mailtolink> Configuration Element

This element allows for the specification of a single hyperlink that the end users of Ephox EditLive! for XML will be provided with via the **Insert Hyperlink** dialog.

Configuration Element Tree Structure

- `<editLive>`
  - `<hyperlinks>`

- `<mailtoList>`
  - **`<mailtolink>`**

```
<editLive>
     ...
     <hyperlinks>
          ...
          <mailtoList>
               <mailtolink ... />
          </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! for XML.

Example

The following example demonstrates how to specify an e-mail address to provide the end users of EditLive! for XML with.

```
<editLive>
    ...
    <hyperlinks>
        ...
        <mailtoList>
            <mailtolink href="someone@mailserver.com"
```

```
                        description="This is a mailto link" />
        </mailtoList>
    </hyperlinks>
    ...
</editLive>
```

Remarks

The **`<mailtolink>`** element can appear multiple times within the **`<mailtoList>`** element.

The **`<mailtolink>`** 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:

```
<mailtolink href=... />
```

<menuBar> Configuration Element

The configuration information contained within this element contains the various settings to use for the menu bar within Ephox EditLive! for XML.

Items will appear in the menu bar of EditLive! for XML, from left to right, in the order that they appear in the EditLive! for XML configuration document.

Note

The configuration information within this element includes settings for each menu (eg. **View**, **Edit**).

Configuration Element Tree Structure

- `<editLive>`
    - **`<menuBar>`**

```
<editLive>
     ...
     <menuBar>
```

```
          <!--menu bar configuration settings-->
     </menuBar>
     ...
</editLive>
```

Optional Attributes

showAboutMenu	This attribute is a boolean which indicates if the Ephox logo branding and associated **About** menu item should be placed on the menu bar. When set to `false` the Ephox logo is removed from the menu. *Default Value:* `true`

Child Elements

<menu>	This element contains the settings for a specific menu (eg. **View**, **Edit**).

Example

The following example demonstrates how to remove the Ephox logo from the menu bar.

```
<editLive>
  ...
  <menuBar showAboutMenu="false">
    ...
  </menuBar>
  ...
</editLive>
```

Remarks

The **`<menuBar>`** element can appear only once within the **`<editLive>`** element.

<menu> Configuration Element

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 (eg. **Edit**, **View**).

Child Elements

<menuItem>	This element contains information for an item on the menu (eg. Cut, Undo, Table Properties).
<designerMenuItem>	This element specifies an item to include within a menu in the Visual Designer.
<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 XML. For example, the **Source View** and **Design View** commands exist in a menuItemGroup.
<menuSeparator>	This element informs EditLive! for XML 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 Ephox EditLive! for XML.
<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.

Example

The following example demonstrates how to create a menu called Edit.

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

Remarks

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

<menuItem> Configuration Element

This element specifies an item to include within a menu in Ephox EditLive! for XML.

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 one of the following:

- New - The **New** command.
- Open - The **Open...** command.
- Save - The **Save** command.
- SaveAs - The **Save As...** command.
- Undo - The **Undo** command.
- Redo - The **Redo** command.
- Cut - The **Cut** command.
- Paste - The **Paste** command.
- SelectAll - The **Select All** command.

- Find - The **Find...** command.
- HLink - The **Insert Hyperlink...** command.
- HRule - The **Insert Horizontal Rule** command.
- Symbol - The **Insert Symbol...** command.
- Bookmark - The **Insert Bookmark...** command.
- ImageLocal - The **Insert Local Image...** command.
- ImageServer - The **Insert Server Image...** command.
- InsTable - The **Insert Table...** command.
- InsRowCol - The **Insert Row or Column** command.
- InsCell - The **Insert Cell** command.
- DelRow -The **Delete Row** command.
- DelCol - The **Delete Column** command.
- DelCell - The **Delete Cell** command.
- Split - The **Split Cell...** command.
- Merge - The **Merge Cells** command.
- PropCell - The **Cell Properties...** command.
- PropTable - The **Table Properties...** command.
- Gridlines - The **Show Gridlines** command.
- Spelling - The **Spelling...** command.
- WordCount - The **Word Count...** command.
- Color - The (text) **Color** command.
- Bold - The **Bold** command.
- Italic - The **Italic** command.
- Underline - The **Underline** command.

- IncreaseIndent - The **Increase Indent** command.
- DecreaseIndent - The **Decrease Indent** command.
- PropRow - The **Row Properties...** command.
- PropCol - The **Column Properties...** command.
- HighlightColor - The **Highlight Color** command.
- Strike - The **Strikethrough** command.
- RemoveFormatting - The **Remove Formatting** command.
- PropList - The **List Properties...** command.
- PropImage - The **Image Properties...** command.
- EditTag - The **Edit Custom Tag...** command.

The following items may only be used within EditLive! for XML

- xmlInsertBefore - The **Insert Before** command.
- xmlInsertAfter - The **Insert After** command.
- xmlInsertAtCurrent - The **Insert Into** command.
- xmlConvert - The **Convert Element** command.
- xmlAddAttribute - The **Add Attribute** command.
- xmlMoveUp - The **Move Up** command.
- xmlMoveDown - The **Move Down** command.
- xmlRemove - The **Remove** command.
- ShowDocumentNavigator - The **Document Navigator** command.
- ShowValidationPane - The **Validation Pane** command.
- xmlSelect - The **Select** (element) command.

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.
mnemonic	This attribute is a single letter which provides the mnemonic for the menu item.

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! for XML 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="Copy"/>
            <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>** Element

<designerMenuItem> Configuration Element

This element specifies an item to include within a menu or the shortcut menu in the Visual Designer.

Configuration Element Tree Structure

- `<editLive>`
  - `<menuBar>`

    - `<menu>`
      - **`<designerMenuItem>`**

OR

- `<editLive>`
  - `<shortcutMenu>`
    - `<shrtMenu>`
      - **`<designerMenuItem>`**

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

Required Attributes

name This attribute gives the name for the designer menu item. For use within a **`<menu>`** element it must be one of the following:

- ControlProperties - The **Control Properties...** command.
- ExportXSLT - The **Export StyleSheet** command

- ExportXSD - The **Export Schema** command.

Examples

The following example demonstrates how to add the FieldProperties, ExportXSLT and ExportXSD items to the **Designer** menu. Thus the instance of the Visual Designer using this configuration will have only a **Designer** menu with these items.

```
<editLive>
    ...
    <menuBar>
        <menu name="Designer">
            <designerMenuItem name="FieldProperties"/>
            <designerMenuItem name="ExportXSLT"/>
            <designerMenuItem name="ExportXSD"/>
        </menu>
    </menuBar>
    ...
</editLive>
```

Remarks

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

The **`<designerMenuItem>`** 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:

```
<designerMenuItem name=... />
```

<menuItemGroup> Configuration Element

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 XML. For example, the **Source View** and **Design View** commands exist in a **`<menuItemGroup>`**.

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 one of the following:

- SourceView - The **Design View** and **HTML View** commands.
- FrameView - The **Browser View** and **Window View** commands.
- List - The **Ordered List** and **Unordered List** commands.
- Align - The **Align Left**, **Align Center** and **Align Right** commands.
- Script - The **Subscript** and **Superscript** commands.

Examples

The following example demonstrates how to add the **Design View** and **HTML View** commands to the menu bar.

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

Remarks

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

The **`<menuGroupItem>`** 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:

```
<menuGroupItem name=... />
```

<menuSeparator> Configuration Element

This element places a horizontal separating line between commands within a Ephox EditLive! for XML menu bar menu. This line will appear between the commands defined by the **`<menuItem>`** elements immediately before and after the **`<menuSeparator>`** element.

Note

This element has no attributes or child elements.

Configuration Element Tree Structure

- `<editLive>`
    - `<menuBar>`

- `<menu>`
    - **`<menuSeparator>`**

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

Examples

The following example demonstrates how to insert a menu separator into a menu. In the instance of EditLive! for XML 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 />
```

<customMenuItem> Configuration Element

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

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 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 given name
- `customPropertiesDialog` - Call a JavaScript function with the given name and pass it the current tag's properties
- `PostDocument` - Post the content of the applet to a server side script

value	The value of the text or hyperlink to be inserted or the name of the JavaScript function to be called when this menu item is clicked.

Note

When using the `insertHTMLAtCursor` action the HTML to be inserted must be URL encoded [http://www.ietf.org/rfc/rfc2396.txt?number=2396] in the XML 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.

Note

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

xhtmlonly	This attribute defines whether the custom menu item should be active only when the cursor is placed within an XHTML section. Setting this attribute to `true` will ensure that the menu item is only active when the cursor is within an XHTML section. *Default:* `false`
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! for XML. The menu 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>
    ...
    <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! for XML. 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 [http://www.ietf.org/rfc/rfc2396.txt?number=2396] as it is in the example above.

See Also

- URL Encoding (RFC 2396) [http://www.ietf.org/rfc/rfc2396.txt?number=2396]
- Raising a JavaScript Event from Ephox EditLive! for XML
- Custom Properties Dialogs for EditLive! for XML
- Submitting EditLive! for XML Content Directly Via HTTP POST
- Custom Menu and Toolbar Items for EditLive! for XML

<submenu> Configuration Element

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

Required Attributes

name This attribute specifies the name of the submenu. It must be one of the following:

- mnuFontFace - The **Font** submenu.
- mnuFontSize - The **Size** submenu.
- mnuStyle - The **Style** submenu.
- mnuColor - The **Color** submenu. This submenu should only be used when

customizing the **Color** submenu as detailed in the section on Customizing Color Choosers.

- mnuHighlightColor - The **Highlight Color** submenu. This submenu should only be used when customizing the **Highlight Color** submenu as detailed in the section on Customizing Color Choosers.

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.

See Also

- Customizing Color Choosers

<toolbars> Element

This element contains the configuration information for the toolbars within Ephox EditLive! for XML. This includes the **Format** and the **Command** toolbars and the buttons and combo boxes contained within them.

Element Tree Structure

- `<editLive>`
  - **`<toolbars>`**

```
<editLive>
     ...
     <toolbars>
          <!--toolbars configuration settings-->
     </toolbars>
     ...
</editLive>
```

Child Elements

<toolbar>	This element contains the configuration information for a toolbar within EditLive! for XML.

Remarks

The **`<toolbars>`** element can appear only once within the **`<editLive>`** element.

<toolbar> Configuration Element

This element contains the configuration information for a toolbar for use within Ephox EditLive! for XML.

Items will appear in a toolbar in EditLive! for XML, from left to right, in the order that they appear in the EditLive! for XML 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 collection of toolbars for EditLive! for XML.

Child Elements

<toolbarButton>	This element will cause a particular button to be present on the toolbar within EditLive! for XML.
<toolbarButtonGroup>	This element will cause a particular group of buttons to be present on the toolbar within EditLive! for XML. The operation of these buttons within EditLive! for XML 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! for XML.
<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 Ephox EditLive! for XML.
<customToolbarComboBox>	This element specifies the properties for a developer defined custom toolbar combo box for use within Ephox EditLive! for XML.

Example

This example demonstrates how to declare toolbars with the name `command` and `format`.

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

Remarks

Toolbars will appear in the EditLive! for XML interface in the order that they appear in the configuration file.

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

<toolbarButtonGroup> Element

This element will cause a particular group of buttons to be present on the toolbar within Ephox EditLive! for XML. 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! for XML 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**.

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. It must be one of the following:

- Align - The **Align Left**, **Align Center** and **Align Right** buttons.
- List - The **Ordered List** and **Unordered List** buttons.
- Script - The **Subscript** and **Superscript** buttons.

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=... />
```

<toolbarSeparator> Configuration Element

This element will cause the appearance of a vertical separating line between toolbar elements in Ephox EditLive! for XML.

Note

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 />
```

<toolbarButton> Element

This element will cause a particular button to be present on the toolbar within Ephox EditLive! for XML.

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. It must be one of the following:

- Cut - The **Cut** button.
- Copy - The **Copy** button.
- Paste - The **Paste** button.
- Find - The **Find** button.
- Undo - The **Undo** button.
- Redo - The **Redo** button.
- HRule - The **Insert Horizontal Rule** button.
- HLink - The **Hyperlink** button.
- Symbol - The **Insert Symbol** button.
- Bookmark - The **Insert Bookmark** button.
- ImageServer - The **Insert Server Image** button.
- InsTable - The **Insert Table** button.
- InsRow - The **Insert Row** button.
- InsCol - The **Insert Column** button.
- DelRow - The **Delete Row** button.
- DelCol - The **Delete Column** button.
- Split - The **Split Cell** button.
- Merge - The **Merge Cells** button.
- Gridlines - The **Show Gridlines** button.
- Bold - The **Bold** button.
- Italics - The **Italics** button.
- Underline - The **Underline** button.
- Color - The (text) **Color** button.

- Spelling - The **Check Spelling** button.
- IncreaseIndent - The **Increase** Indent button.
- DecreaseIndent - The **Decrease Indent** button.
- New - The **New** button.
- Open - The **Open...** button.
- Save - The **Save** button.
- Strike - The **Strikethrough** button.
- RemoveFormatting - The **Remove Formatting** button.
- HighlightColor - The **Highlight Color** button.
- WordCount - The **Word Count** button.

The following items may only be used within EditLive! for XML

- xmlMoveUp - The **Move Up** command.
- xmlMoveDown - The **Move Down** command.
- xmlValidate - The **Validate XML** command.

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=... />
```

<toolbarComboBox> Element

This element will cause a particular combo box to be present on the toolbar within Ephox EditLive! for XML.

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 button group. It must be one of the following:

- tlbStyle - The **Style** combo box containing the various styles available to end users (eg. `Heading 1`, `Heading 2`).
- tlbFace - The **Typeface** combo box containing the various typefaces or fonts available to end users (eg. `Times New Roman`, `Arial`).
- tlbSize - The **Size** combo box containing the various font sizes available to end users (eg. `12pt`, `14pt`).
- tlbColor - The **Color** drop down. This drop down should only be used when customizing the **Color** drop down as detailed in the section on Customizing Color Choosers.
- tlbHighlightColor - The **Highlight Color** drop down. This drop down should only be used when customizing the **Highlight Color** drop down as detailed in the section on Customizing Color Choosers.

Child Elements

<comboBoxItem>	This element contains the information required by EditLive! for Java to configure an item within one of the EditLive! for XML 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.

See Also

- Customizing Color Choosers

<comboBoxItem> Configuration Element

This element contains the information required by Ephox EditLive! for XML to configure an item within one of the EditLive! for XML 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! for Java.

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 `<FONT>` tag used within the EditLive! for Java HTML source code.

Example: If the name attribute was set to Times New Roman then the following `<FONT>` tag would be inserted into the EditLive! for XML HTML source code:

```
<FONT face="Times New Roman">
```

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 `<FONT>` tag used within the EditLive! for Java HTML source code.

Note

The **name** attribute for the **Size** combo box must be between 1 and 7, inclusive.

Example: If the name attribute was set to 3 then the following `<FONT>` tag would be inserted into the EditLive! for XML HTML source code:

```
<FONT size="3">
```

Optional Attributes

text	This attribute gives the value which appears inside the relevant combo box within EditLive! for XML (eg. Heading 1, Normal, 12pt, Times New Roman).

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! for XML. Also added is the Arial font to the **Typeface** combo box and it is listed as "Company Font" in the combo box in EditLive! for XML. Finally the HTML font size 3 is added to the **Size** combo box and lists it as "12pt" in the combo box in EditLive! for XML.

All the combo boxes in this example are added to the **Format Toolbar**.

```
<editLive>
    ...
    <toolbars>
        <toolbar name="format">
            <toolbarComboBox name="Style">
                <comboBoxItem name="H1" text="Heading 1"/>
            </toolbarComboBox>
            <toolbarComboBox name="Face">
                <comboBoxItem name="Arial" text="Company Font"/>
            </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=... />
```

<customToolbarButton> Configuration Element

This element will cause a particular button to be present on the toolbar within Ephox EditLive! for XML.

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.

> **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 given name
> - `customPropertiesDialog` - Call a JavaScript function with the given name and pass it the current tag's properties
> - `PostDocument` - Post the content of the applet to a server side script

value The value of the text or hyperlink to be inserted or the name of the JavaScript function to be called when this toolbar button is clicked.

Note

When using the `insertHTMLAtCursor` action the HTML to be inserted must be URL encoded [http://www.ietf.org/rfc/rfc2396.txt?number=2396] in the XML 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 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. **Note** This URL can be relative or absolute. If relative the URL is relative to the URL of the page in which EditLive! for XML is embedded.
xhtmlonly	This attribute defines whether the custom toolbar button should be active only when the cursor is placed within an XHTML section. Setting this attribute to `true` will ensure that the toolbar button is only active when the cursor is within an XHTML section. *Default:* `false`
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).

Example

The following example demonstrates how to define a custom toolbar button for use within EditLive! for XML on the **Command Toolbar**. 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="command">
            <customToolbarButton
                name="customButton1"
                text="Custom Button"
                imageURL="http://www.someserver.com/image20x20.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 for use within EditLive! for XML on the **Format Toolbar**. The button defined in this example is used with a custom properties dialog. The custom properties dialog will be available for use when the cursor is inside any `<td>` tag.

```
<editLive>
  ...
  <toolbars>
    <toolbar name="format">
      <customToolbarButton
        name="customPropButton1"
        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>`**.

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 [http://www.ietf.org/rfc/rfc2396.txt?number=2396] as it is in the example above.

See Also

- URL Encoding (RFC 2396) [http://www.ietf.org/rfc/rfc2396.txt?number=2396]
- Raising a JavaScript Event from Ephox EditLive! for XML
- Custom Properties Dialogs for EditLive! for XML
- Submitting EditLive! for XML Content Directly Via HTTP POST
- Custom Menu and Toolbar Items for EditLive! for XML

<customToolbarComboBox> Configuration Element

This element specifies the properties for a developer defined custom toolbar combo box for use within Ephox EditLive! for XML.

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.

Optional Attributes

xhtmlonly	This attribute defines whether the custom toolbar combo box should be active only when the cursor is placed within an XHTML section. Setting this attribute to `true` will ensure that the toolbar combo box is only active when the cursor is within an XHTML section. *Default:* `false`

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! for XML 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.

<customComboBoxItem> Configuration Element

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

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.

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 given name
- `customPropertiesDialog` - Call a JavaScript function with the given name and pass it the current tag's properties
- `PostDocument` - Post the content of the applet to a server side script

value	The value of the text or hyperlink to be inserted or the name of the JavaScript function to be called when this toolbar button is clicked.

Note

When using the `insertHTMLAtCursor` action the HTML to be inserted must be URL encoded [http://www.ietf.org/rfc/rfc2396.txt?number=2396] in the XML file. For example, `<p>HTML to insert<p>` becomes `%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 **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=... />
```

Text assigned to the **value** attribute must be URL encoded [http://www.ietf.org/rfc/rfc2396.txt?number=2396] as it is in the example above.

See Also

- URL Encoding (RFC 2396) [http://www.ietf.org/rfc/rfc2396.txt?number=2396]
- Raising a JavaScript Event from Ephox EditLive! for XML
- Custom Properties Dialogs for EditLive! for XML

- Submitting EditLive! for XML Content Directly Via HTTP POST
- Custom Menu and Toolbar Items for EditLive! for XML

<shortcutMenu> Configuration Element

This element contains the configuration information for the shortcut menu within Ephox EditLive! for XML.

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 Ephox EditLive! for XML.
`<elementMenu>`	This element allows for the configuration of the element menu associated with the Document Navigator bar in EditLive! for XML.

Note

This element is only applicable when creating a configuration file for EditLive! for XML.

Remarks

The **`<shortcutMenu>`** element can appear only once within the **`<editLive>`** element.

<shrtMenu> Configuration Element

This element allows for the configuration of the shortcut menu within Ephox EditLive! for XML. 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.
<designerMenuItem>	This element specifies an item to include within the shortcut menu in the Visual Designer.
<shrtMenuSeparator>	This element will cause the appearance of a horizontal separating line between shortcut menu commands.

Remarks

The **`<shrtMenu>`** element can appear only once within the **`<shortcutMenu>`** element.

<shrtMenuItem> Configuration Element

This element defines command items for the shortcut menu within Ephox EditLive! for XML.

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 shortcut menu command. It must be one of the following:

- New - The **New** command.
- Open - The **Open...** command.

- Save - The **Save** command.
- SaveAs - The **Save As...** command.
- Undo - The **Undo** command.
- Redo - The **Redo** command.
- Cut - The **Cut** command.
- Paste - The **Paste** command.
- SelectAll - The **Select All** command.
- Find - The **Find...** command.
- HLink - The **Insert Hyperlink...** command.
- HRule - The **Insert Horizontal Rule** command.
- Symbol - The **Insert Symbol...** command.
- Bookmark - The **Insert Bookmark...** command.
- ImageLocal - The **Insert Local Image...** command.
- ImageServer - The **Insert Server Image...** command.
- InsTable - The **Insert Table...** command.
- InsRowCol - The **Insert Row or Column** command.
- InsCell - The **Insert Cell** command.
- DelRow -The **Delete Row** command.
- DelCol - The **Delete Column** command.
- DelCell - The **Delete Cell** command.
- Split - The **Split Cell...** command.
- Merge - The **Merge Cells** command.
- PropCell - The **Cell Properties...** command.
- PropTable - The **Table Properties...** command.

- Gridlines - The **Show Gridlines** command.
- Spelling - The **Spelling...** command.
- WordCount - The **Word Count...** command.
- Color - The (text) **Color** command.
- Bold - The **Bold** command.
- Italic - The **Italic** command.
- Underline - The **Underline** command.
- IncreaseIndent - The **Increase Indent** command.
- DecreaseIndent - The **Decrease Indent** command.
- PropRow - The **Row Properties...** command.
- PropCol - The **Column Properties...** command.
- HighlightColor - The **Highlight Color** command.
- Strike - The **Strikethrough** command.
- RemoveFormatting - The **Remove Formatting** command.
- PropList - The **List Properties...** command.
- PropImage - The **Image Properties...** command.
- EditTag - The **Edit Custom Tag...** command.
- Style - The **Style** submenu.
- FontFace - The **Font** submenu.
- Size - The **Size** submenu.
- Reformat - The **Reformat Code** command.

The following items may only be used within EditLive! for XML

- xmlInsertBefore - The **Insert Before** command.
- xmlInsertAfter - The **Insert After** command.

- xmlInsertAtCurrent - The **Insert Into** command.
- xmlConvert - The **Convert Element** command.
- xmlAddAttribute - The **Add Attribute** command.
- xmlMoveUp - The **Move Up** command.
- xmlMoveDown - The **Move Down** command.
- xmlRemove - The **Remove** command.
- ShowDocumentNavigator - The **Document Navigator** command.
- ShowValidationPane - The **Validation Pane** command.
- xmlSelect - The **Select** (element) command.

Example

The following example configures the **Shortcut Menu** of EditLive! for XML 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> Configuration Element

This element places a horizontal separating line between commands within the Ephox EditLive! for XML shortcut menu. This line will appear between the commands defined by the **`<shrtMenuItem>`** elements immediately before and after the **`<shrtMenuSeparator>`** element.

Note

This element has no attributes or child elements.

Configuration Element Tree Structure

- `<editLive>`
  - `<shortcutMenu>`
    - `<shrtMenu>`

      OR

      `<elementMenu>`
      - **`<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! for XML 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 />
```

<elementMenu> Configuration Element

This element allows for the configuration of the element menu associated with the **Document Navigator Bar** in EditLive! for XML. It is the child element of the **`<shortcutMenu>`** element.

Note

This element only applies when configuring EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - `<shortcutMenu>`
    - **`<elementMenu>`**

```
<editLive>
     ...
     <shortcutMenu>
          <elementMenu>
               <!--element menu configuration settings-->
          </element>
     </shortcutMenu>
</editLive>
```

Child Elements

<elementMenuItem>	This element defines command items for the element menu associated with the **Document Navigator Bar** within Ephox EditLive! for XML.
<shrtMenuSeparator>	This element will cause the appearance of a horizontal separating line between element menu commands.

Remarks

The **`<elementMenu>`** element can appear only once within the **`<shortcutMenu>`** element.

<elementMenuItem> Configuration Element

This element defines command items for the element menu associated with the **Document Navigator Bar** within Ephox EditLive! for XML.

Note

This element only applies when configuring EditLive! for XML.

Configuration Element Tree Structure

- `<editLive>`
  - `<shortcutMenu>`
    - `<elementMenu>`
      - **`<elementMenuItem>`**

```
<editLive>
     ...
     <shortcutMenu>
          <elementMenu>
               <elementMenuItem ... />
          </elementMenu>
     </shortcutMenu>
     ...
</editLive>
```

Required Attributes

name This attribute gives the name for the shortcut menu command. It must be one of the following:

- New - The **New** command.
- Open - The **Open...** command.

- Save - The **Save** command.
- SaveAs - The **Save As...** command.
- Undo - The **Undo** command.
- Redo - The **Redo** command.
- Cut - The **Cut** command.
- Paste - The **Paste** command.
- SelectAll - The **Select All** command.
- Find - The **Find...** command.
- HLink - The **Insert Hyperlink...** command.
- HRule - The **Insert Horizontal Rule** command.
- Symbol - The **Insert Symbol...** command.
- Bookmark - The **Insert Bookmark...** command.
- ImageLocal - The **Insert Local Image...** command.
- ImageServer - The **Insert Server Image...** command.
- InsTable - The **Insert Table...** command.
- InsRowCol - The **Insert Row or Column** command.
- InsCell - The **Insert Cell** command.
- DelRow -The **Delete Row** command.
- DelCol - The **Delete Column** command.
- DelCell - The **Delete Cell** command.
- Split - The **Split Cell...** command.
- Merge - The **Merge Cells** command.
- PropCell - The **Cell Properties...** command.
- PropTable - The **Table Properties...** command.

- Gridlines - The **Show Gridlines** command.
- Spelling - The **Spelling...** command.
- WordCount - The **Word Count...** command.
- Color - The (text) **Color** command.
- Bold - The **Bold** command.
- Italic - The **Italic** command.
- Underline - The **Underline** command.
- IncreaseIndent - The **Increase Indent** command.
- DecreaseIndent - The **Decrease Indent** command.
- PropRow - The **Row Properties...** command.
- PropCol - The **Column Properties...** command.
- HighlightColor - The **Highlight Color** command.
- Strike - The **Strikethrough** command.
- RemoveFormatting - The **Remove Formatting** command.
- PropList - The **List Properties...** command.
- PropImage - The **Image Properties...** command.
- EditTag - The **Edit Custom Tag...** command.
- Style - The **Style** submenu.
- FontFace - The **Font** submenu.
- Size - The **Size** submenu.
- ShowDocumentNavigator - The **Document Navigator** command.
- ShowValidationPane - The **Validation Pane** command.
- Reformat - The **Reformat Code** command.
- xmlAddAttribute - The **Add Attribute** command.

- xmlInsertBefore - The **Insert Before** command.
- xmlInsertAfter - The **Insert After** command.
- xmlInsertAtCurrent - The **Insert Into** command.
- xmlConvert - The **Convert Element** command.
- xmlMoveUp - The **Move Up** command.
- xmlMoveDown - The **Move Down** command.
- xmlRemove - The **Remove** command.
- xmlSelect - The **Select** (element) command.

Example

The following example configures the **Shortcut Menu** of EditLive! for XML so that it contains the **Insert Before**, **Insert After**, **Insert Into**, **Convert Element**, **Select**, and **Remove** commands.

```
<editLive>
    ...
    <shortcutMenu>
        <elementMenu>
          <elementMenuItem name="xmlInsertBefore"/>
          <elementMenuItem name="xmlInsertAfter"/>
          <elementMenuItem name="xmlInsertAtCurrent"/>
          <elementMenuItem name="xmlConvert"/>
          <elementMenuItem name="xmlSelect"/>
          <shrtMenuSeparator/>
          <elementMenuItem name="xmlRemove"/>
        </elementMenu>
    </shortcutMenu>
</editLive>
```

Remarks

It is recommended that element menu be configured so that it contains only items with functionality relating to XML elements such as **Insert Before** and **Convert Element**.

The **`<elementMenuItem>`** element can appear multiple times within the **`<elementMenu>`** element.

The **`<elementMenuItem>`** 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:

```
<elementMenuItem name=... />
```

Index