

# XStandard Developer's Guide: Table Of Contents

This guide is intended for use with XStandard version 2.0. Please refer to [Changes From Previous Version](#) for notes on the differences between this guide and the previous version.

- [Requirements](#)
- [Interface](#)
- [Features](#)
- [Architecture](#)
- [Installation](#)
- [API Reference](#)
- [Web Integration](#)
  - [Step 1](#)
  - [Step 2](#)
  - [Step 3](#)
  - [Step 4](#)
  - [Step 5](#)
  - [Examples](#)
  - [Integration FAQs](#)
- [App Integration](#)
  - [Visual Studio](#)
  - [Access](#)
  - [Visual Basic 6](#)
  - [Visual C++ 6](#)
  - [Delphi 7](#)
  - [Visual FoxPro 9](#)
- [Accessibility](#)
- [Localization](#)
- [Web Services](#)
  - [Spell Checker](#)
  - [Image Library & Attachment Library](#)
  - [Directory](#)
  - [Subdocument](#)
- [Toolbar Customization](#)
  - [Styles](#)
  - [Buttons](#)
- [Best Practices](#)
- [Advanced Topics](#)
  - [Caching](#)
  - [Heartbeat](#)
  - [Placeholders](#)
  - [Browser Preview Customization](#)
  - [Screen Reader Preview Customization](#)
  - [Namespaces](#)
  - [Locking](#)
  - [Markers](#)
- [License File](#)
- [Did You Know?](#)
- [Changes From Previous Version](#)
- [Copyright](#)

## Requirements

- [Client Requirements](#)
- [Server Requirements](#)
- [Current Release](#)

## Client Requirements

For browser-based applications

IE 5+, Firefox 1.0+, Safari 1.3+, Opera 9.0+

#### For desktop applications

Visual Studio, Access, VB 6, VC++, Visual FoxPro, Delphi

#### Operating Systems

Windows 2000, XP, Vista, Windows 7 or Windows 8

## Server Requirements

When used in Web applications, XStandard can work with *any* server-side scripting environment such as ASP, ASP.NET, PHP, ColdFusion, JSP, etc. XStandard Pro comes with server-side software called Web Services for file uploading, building image libraries and spell checking. Currently, this software is available for ASP and ASP.NET on Windows and PHP on Windows/Linux/FreeBSD.

## Current Release

### Version

3.0

### Download Packages

- x-lite.exe (1.9 MB) XStandard Lite for Windows
- x-pro.exe (2.0 MB) XStandard Pro for Windows

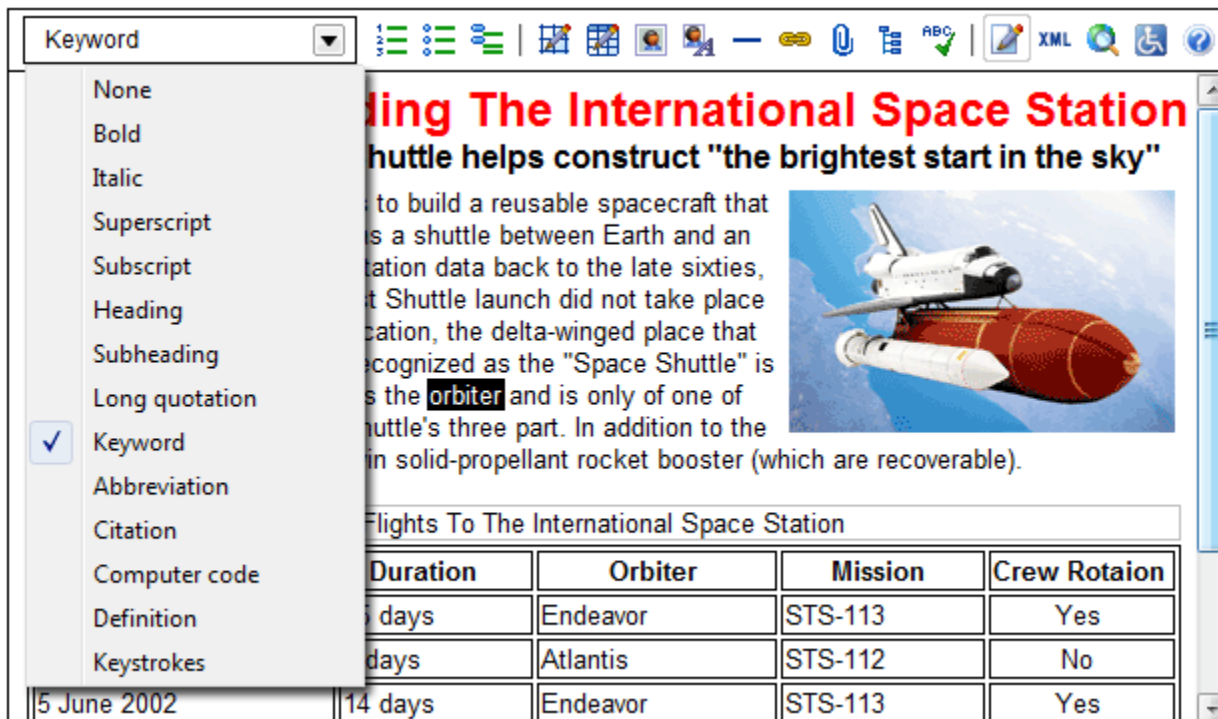
[Download XStandard](#)

## Interface

- [Toolbar](#)
  - [Edit Mode](#)
  - [View Source Mode](#)
  - [Browser Preview Mode](#)
  - [Screen Reader Preview Mode](#)
- [Context Menu](#)

## Toolbar

The editor's toolbar is the primary means of accessing the editor's functionality. The screen shot below shows XStandard with the drop-down Styles menu extended on the toolbar.



The screenshot shows the XStandard editor interface. At the top is a toolbar with various icons for editing and formatting. A drop-down menu is open, showing a list of styles. The 'Keyword' style is selected, indicated by a checkmark. The main editing area displays a document with a title 'Building The International Space Station' in red, followed by a paragraph of text and an image of a space shuttle. Below the text is a table with columns for Duration, Orbiter, Mission, and Crew Rotaion.

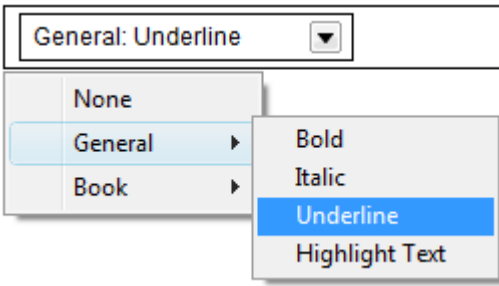
Duration	Orbiter	Mission	Crew Rotaion
14 days	Endeavor	STS-113	Yes
14 days	Atlantis	STS-112	No
15 June 2002	Endeavor	STS-113	Yes

XStandard's toolbar is flexible and highly customizable. Buttons can be shown, hidden or re-arranged. Existing icons can be modified or new buttons added. See the [Toolbar Customization](#) section of this document for instructions. If necessary, XStandard's toolbar can also be hidden entirely and replaced by a different toolbar that communicates with the editor through its [API](#).














The editor has 4 modes of operation (Edit, View Source, Browser Preview and Screen Reader Preview). Different toolbar buttons are available on the toolbar in each mode, as described in the following sections.

## Edit Mode





### The Drop-down Styles Menu

Toolbar Function	Description
	<p><b>Styles menu</b> - Business users format content by selecting from a drop-down menu of style options. The Styles Menu list is customizable and styling options typically reflect the type of document being edited. This greatly simplifies the authoring process and ensures consistency of presentation.</p> <p>User-friendly names in the Styles menu carry instructions for the type of element and attribute(s) to be created. In our example, selecting "Bold" creates a <code>&lt;strong&gt;</code> tag. Selecting "Underline" creates a span tag with a class called "underline": <code>&lt;span class="underline"&gt;</code>.</p> <p>Styles appearing in the Styles Menu are defined in an XML document that the editor reads when it is started. More information is available under the <a href="#">Styles</a> section.</p>



















### Default Toolbar Buttons In Edit Mode

Toolbar Function	ID	Description
	ordered-list	<b>Numbered list</b> - This button creates an ordered list of sequentially numbered items.
	unordered-list	<b>Bulleted list</b> - This button creates an unordered list of bulleted items.
	definition-list	<b>Definition list</b> - This button creates a definition list which can be used to author a glossary of terms, to present dialog, or other associated content.
	draw-layout-table	<b>Draw layout table</b> - This button is used to graphically specify the dimensions of a layout table.
	draw-data-table	<b>Draw data table</b> - This button is used to graphically specify the dimensions of a data table.
	image	<b>Image</b> - This button is used to insert images. Users can enter a specific URL where the image can be found, browse a library of images, or select an image from their local computer.
	images-as-text	<b>Show images as text</b> - This button displays alternate text in place of images so that alternate text can be seen and edited in the document.
	separator	<b>Separator</b> - This button inserts a content separator. By default, this renders as a line.
	hyperlink	<b>Hyperlink</b> - This button is used to insert or edit hyperlinks (or anchors). The button is active only when text or an image is selected.
	attachment	<b>Attachment</b> - This button is available in the Pro version of XStandard, and is used to browse attachment libraries. The button is active only when text or an image is selected.
	directory	<b>Directory</b> - This button is available in XStandard Pro. It is used to browse third-party systems such as a CMS and inserts code snippets into the editor.
	spellchecker	<b>Spelling</b> - This button spell-checks the contents of the editor and is available in XStandard Pro.
	wysiwyg	<b>Edit</b> - Changes the editor's view to Edit (WYSIWYG) mode.

















## Default Toolbar Buttons In Edit Mode

Toolbar Function	ID	Description
	source	<b>View source</b> - This button displays the XHTML source code of content managed by the editor.
	preview	<b>Browser preview</b> - This presents the editor's content as IE would display it.
	screen-reader	<b>Screen reader preview</b> - This displays content managed through the editor as a screen reader would process it, in linear fashion. This feature gives authors opportunities to review and optimize content for greater accessibility.
	help	<b>Help</b> - Opens a new window containing end-user documentation for the editor.

## Additional Toolbar Buttons In Edit Mode







Toolbar Function	ID	Description
	strong	<b>Bold</b>
	em	<b>Italic</b>
	cut	<b>Cut</b>
	copy	<b>Copy</b>
	paste	<b>Paste</b>
	sub	<b>Subscript</b>
	sup	<b>Superscript</b>
	underline	<b>Underline</b> - By default, this button creates the markup: <code>&lt;span class="underline"&gt;</code> . This markup can be customized.
	strikethrough	<b>Strikethrough</b> - By default, this button creates the markup: <code>&lt;span class="strikethrough"&gt;</code> . This markup can be customized.
	undo	<b>Undo</b>
	redo	<b>Redo</b>
	blockquote	<b>Add long quotation</b> - This button identifies content as a quotation. By default, the text is formatted as a block, and justified to left and right.
	undo-blockquote	<b>Remove long quotation</b> - This removes a long quotation.
	align-left	<b>Align left</b> - By default, this button creates the markup: <code>&lt;p class="left"&gt;</code> . This markup can be customized.
	align-center	<b>Align center</b> - By default, this button creates the markup: <code>&lt;p class="center"&gt;</code> . This markup can be customized.
	align-right	<b>Align right</b> - By default, this button creates the markup: <code>&lt;p class="right"&gt;</code> . This markup can be customized.
	textbox	<b>Text box</b> - By default, this button creates the markup: <code>&lt;div class="textbox"&gt;&lt;h5&gt;{heading}&lt;/h5&gt;&lt;p&gt;{text}&lt;/p&gt;&lt;/div&gt;</code> . This markup can be customized.
	photo	<b>Photo</b> - By default, this button creates the markup: <code>&lt;div class="photo"&gt;&lt;p class="photo"&gt;&lt;img width="100" alt="" height="150" src="images/placeholder.gif" /&gt;&lt;/p&gt;&lt;p class="photo"&gt;{caption/credit}&lt;/p&gt;&lt;/div&gt;</code> . This markup can be customized.

### Additional Toolbar Buttons In Edit Mode




Toolbar Function	ID	Description
	open-document	<b>Open document</b> - This button provides a hook for custom programming by generating an event.
	save	<b>Save document</b> - This button provides a hook for custom programming by generating an event.
	new-document	<b>New document</b> - This button provides a hook for custom programming by generating an event.
	print	<b>Print</b> - This button provides a hook for custom programming by generating an event.
	properties	<b>Properties</b> - This button provides a hook for custom programming by generating an event.
	wizard	<b>Wizard</b> - This button provides a hook for custom programming by generating an event.
	layout-table	<b>Create layout table</b> - This creates a table used for visual layout rather than for presenting data. Layout tables are typically used to arrange images and text to achieve a more pleasing visual effect.
	data-table	<b>Create data table</b> - This button creates a table used for presenting tabular data. An example would be a bus schedule or an expense report. Data tables typically have row and/or column headings and the data inside the table is read and understood in relation to the headings.
	copyright	<b>Copyright symbol</b>
	euro	<b>Euro symbol</b>
	pound	<b>Pound symbol</b>
	registered-trade-mark	<b>Registered trade mark symbol</b>
	trade-mark	<b>Trade mark symbol</b>
	yen	<b>Yen symbol</b>
	expand	<b>Expand</b> - This button is used to open the editor in a larger window. Currently, this functionality is only available in the Windows version of the editor.
	find-replace	<b>Find / replace</b>

### View Source Mode

#### Default Toolbar Buttons In View Source Mode





Toolbar Function	ID	Description
	indent	<b>Indent</b> - This button inserts a TAB character.
	whitespace	<b>Show whitespace</b> - This button toggles the show whitespace feature on/off. When enabled, the editor will render a marker in place of whitespace characters such as soft spaces and tabs.
	word-wrap	<b>Word wrap</b> - This button toggles the word wrap feature on/off.
	dim-tags	<b>Dim tags</b> - This button toggles the dim tags feature on/off. When enabled, the editor will dim or gray out markup characters in order to make it easier to read content.
	validate	<b>Validate</b> - This button is used to check if the markup is well formed according to the rules of XML.
	wysiwyg	<b>Edit</b> - Changes the editor's view to Edit (WYSIWYG) mode.

### Default Toolbar Buttons In View Source Mode

Toolbar Function	ID	Description
	source	<b>View source</b> - This button displays the XHTML source code of content managed by the editor.
	preview	<b>Browser preview</b> - This presents the editor's content as IE would display it.
	screen-reader	<b>Screen reader preview</b> - This displays content managed through the editor as a screen reader would process it, in linear fashion. This feature gives authors opportunities to review and optimize content for greater accessibility.





### Browser Preview Mode

#### Default Toolbar Buttons In Browser Preview Mode

Toolbar Function	ID	Description
	wysiwyg	<b>Edit</b> - Changes the editor's view to Edit (WYSIWYG) mode.
	source	<b>View source</b> - This button displays the XHTML source code of content managed by the editor.
	preview	<b>Browser preview</b> - This presents the editor's content as IE would display it.
	screen-reader	<b>Screen reader preview</b> - This displays content managed through the editor as a screen reader would process it, in linear fashion. This feature gives authors opportunities to review and optimize content for greater accessibility.

### Screen Reader Preview Mode

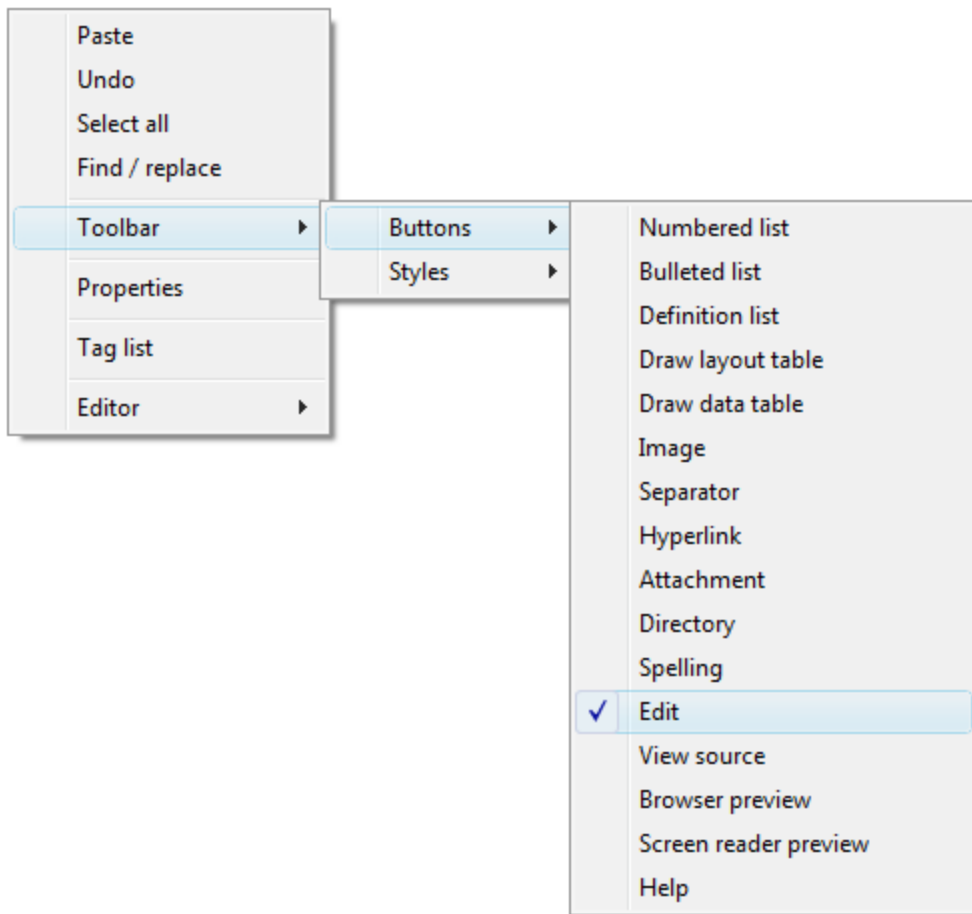
#### Default Toolbar Buttons In Screen Reader Preview Mode

Toolbar Function	ID	Description
	wysiwyg	<b>Edit</b> - Changes the editor's view to Edit (WYSIWYG) mode.
	source	<b>View source</b> - This button displays the XHTML source code of content managed by the editor.
	preview	<b>Browser preview</b> - This presents the editor's content as IE would display it.
	screen-reader	<b>Screen reader preview</b> - This displays content managed through the editor as a screen reader would process it, in linear fashion. This feature gives authors opportunities to review and optimize content for greater accessibility.

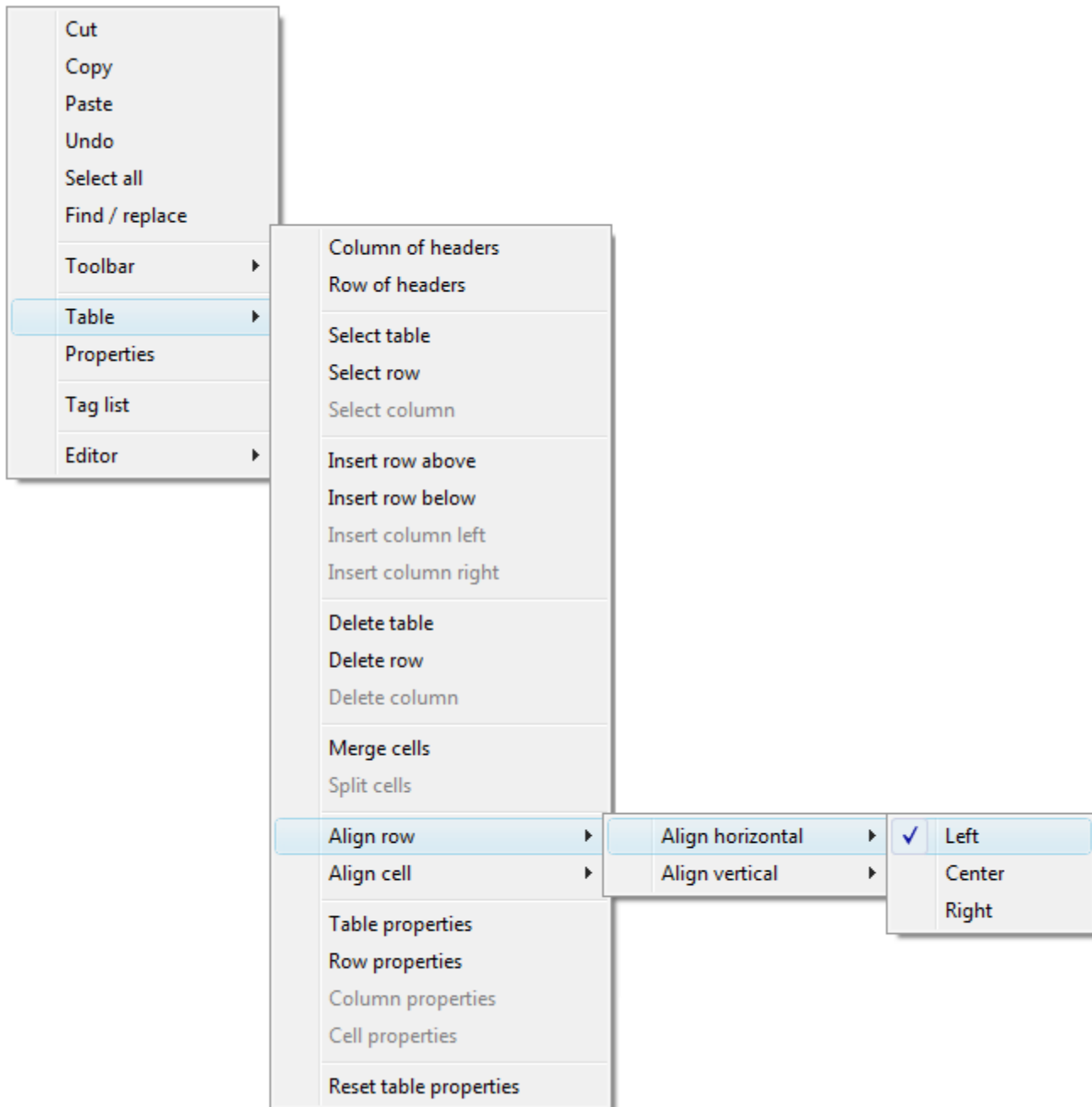
## Context Menu

XStandard's interface provides context-sensitive pop-up menus, or "context menus". These menus are accessed by right mouse clicks, by **SHIFT+F10** on Windows or **CTRL+SPACE** on OS X.

The functionality of the toolbar is fully accessible using a keyboard, through the context menu:



The context sensitive context menus provide access to a lot of functionality . The screen shot below shows table editing features available via the context menu.



## Features





### XStandard: Powerful features. Steered by standards.

This table lists the principal features of XStandard Lite and XStandard Pro, with explanatory screen shots. Hyperlinks lead to additional information. Please [contact us](#) if you have any questions.








XStandard Features				
Feature	Lite	Pro	See It	Notes
Runs in browser-based content management solutions	✓	✓		Internet Explorer, Firefox, Safari and Opera.
Runs in desktop-based content management solutions	✓	✓		Use XStandard wherever ActiveX controls are supported (Visual Basic,





## XStandard Features

Feature	Lite	Pro	See It	Notes
				Visual C++, Access, Visual Studio.NET, Delphi, FoxPro, etc.). See <a href="#">App Integration</a> in this documentation for more details.
Author/edit/spell check alternate text directly in the document	✓	✓		The "Images As Text" feature reduces the skill required to author appropriate alternate text, intuitively clarifies the function of alternate text, and exposes alternate text for the first time to processing by popular editing features such as find/replace and spell checking.
Generates clean XHTML	✓	✓		<a href="#">Why XHTML is best for content management systems (CMS).</a>
Supports most CSS2.1 selectors	✓	✓		<a href="#">Supported CSS 2.1 selectors</a>
A genuine XHTMLeditor, not an HTMLeditor with code clean-up routines.	✓	✓		Most WYSIWYG editors are just JavaScript wrappers around the editing control built into many browsers such as the MSHTML control found in Internet Explorer. These types of editor (which generate HTML and then run code clean-up routines against it) have significant limitations. By contrast, XStandard is built from the ground up to be a true XHTML editor in its own right.
XHTML generated by XStandard can be parsed by XMLparsers.	✓	✓		Use off-the-self XML technologies like DOM, SAX and XSLT to further process markp generated by XStandard. Your CMS can do this before content is saved to the database, or in a batch process. <a href="#">See an example of how to load content generated by XStandard into an XML DOM parser.</a>
Uses Cascading Style Sheets (CSS) for formatting.	✓	✓		XStandard uses external or embedded CSS to ensure data is never fused with formatting. <a href="#">See how to correctly format content.</a>
Makes applying CSSquick and easy	✓	✓		Applying the correct formatting is <a href="#">fast and accurate</a> using XStandard's

## XStandard Features








Feature	Lite	Pro	See It	Notes
				drop-down "Styles" menu that generates the markup that references CSS. User-friendly style names speed the authoring process.
Uses Web Services not FTP for file uploading	✗	✓		<a href="#">Web Services</a> are superior to FTP because they handle metadata and offer tighter integration with content management systems.
Easy-to-use interface	✓	✓		XStandard's streamlined toolbar is a refreshing alternative to the dizzy array of toolbar buttons/controls seen in most WYSIWYG editors. XStandard is able to offer more functionality with a tighter toolbar because advanced functionality is available through the context menu and the Styles drop-down menu. Also, since content in XStandard is formatted through CSS, some toolbar buttons/controls used by other editors are completely unnecessary.
Configurable toolbar	✓	✓		Show / hide / move the buttons on XStandard's toolbar.
Customizable toolbar	✗	✓		<a href="#">Change icons</a> to match the look-and-feel of your own applications. Move frequently used styles to the toolbar. Program buttons to insert code snippets, or use your own buttons to extend XStandard's functionality.
Available in 22 languages	✓	✓		Language versions include: English, French, German, Spanish, Chinese, Dutch, Italian, Russian and Czech.
Create new language versions easily.	✓	✓		Since XStandard stores its localization data in an independent XML file, XStandard's 22 standard interface languages can be easily modified (reworded), or entirely new language versions of XStandard can be created.
Manages content in any language	✓	✓		Author content in multiple languages (including Chinese, Russian, Greek, etc.).

## XStandard Features

Feature	Lite	Pro	See It	Notes
				XStandard is a true Unicode editor.
Imports third-party data	✗	✓		XStandard's timesaving "Directory" feature communicates with third-party applications (such as your CMS) and allows users to insert data from external sources directly into the editor.
Cleans Microsoft Word	✗	✓		XStandard retains structural elements when pasting from Word (lists, tables, hyperlinks, images, headings, etc.), but strips out proprietary Microsoft Office tags and inline formatting.
Has a multi-lingual spell checker and custom dictionary	✗	✓		Spell check in English (US, Canadian, British), German, Danish, Spanish, French, Italian, Dutch, Norwegian, Portuguese and Swedish. Add unusual spellings or frequently used abbreviations to a custom dictionary.
Enables locking of content (read-only)	✗	✓		
Ability to add markers to content	✗	✓		Markers are text labels that can apply short, informative messages to elements of content. For example, they can be used to flag areas of editable or read-only content.
Supports subdocuments	✗	✓		Subdocuments are chunks of reusable content that authors insert into documents as required. Subdocuments are essentially custom elements that act as placeholders for content stored outside the document, within the CMS.
Supports popular table editing options.	✓	✓		Editing options include splitting / merging cells, aligning cell contents (right / left / center / top / middle / bottom), inserting / deleting rows, columns and tables, etc.
Ability to draw tables by dragging	✓	✓		
Supports bullets and numbered lists	✓	✓		
Supports authoring definition lists	✓	✓		XStandard is one of the few editors that support authoring of definition lists. XStandard

## XStandard Features

Feature	Lite	Pro	See It	Notes
				also has unique features that make authoring definition lists easier, including the ability to sort items in the list.
Supports the correct use of block quotes	✓	✓		Most editors use <code>&lt;blockquote&gt;</code> for indenting which is wrong. XStandard encourages the correct use of <code>&lt;blockquote&gt;</code> for quotations only, and uses CSS for indenting.
Supports inline quotes	✓	✓		XStandard is one of the few editors that support the <code>&lt;q&gt;</code> element.
Easily creates links within the current document	✓	✓		XStandard automatically treats document sections (headings <code>h1</code> to <code>h6</code> ) as anchor points and provides an interface to create hyperlinks to these anchors.
Use relative URLs for images	✓	✓		XStandard can be configured to resolve relative URLs in markup so that images with relative URLs can be displayed to users.
Cursor stays in sync when switching between Edit and View Source modes	✓	✓		
Saves images from the editor to the desktop	✓	✓		
Supports drag & drop of image files directly into the editor, as well as file browsing	✗	✓		Drag images from the desktop into the editor. Images will be uploaded to the server. Browse image files on the local computer or in remote libraries. Set limits on file size and type.
Permits entire folders to be dragged directly into the editor	✗	✓		Uploaded folders are automatically zipped and a hyperlink created to the zipped file.
Inserts custom tags	✓	✓		XStandard makes it easy for business users to add semantic meaning to text and objects by inserting <a href="#">custom tags</a> .
Automatically inserts image metadata	✗	✓		Metadata attached to image files in libraries browsed by XStandard is automatically captured when images are selected.

XStandard Features				
Feature	Lite	Pro	See It	Notes
Supports semantic markup tags like <code>&lt;abbr&gt;</code> , <code>&lt;acronym&gt;</code> , <code>&lt;dfn&gt;</code> , <code>&lt;kbd&gt;</code> , <code>&lt;samp&gt;</code> , <code>&lt;code&gt;</code> , <code>&lt;cite&gt;</code> , etc.	✓	✓		Inserts <a href="#">semantic tags</a> that render markup meaningful to visual and non-visual browsers.
Meets or exceeds regional accessibility standards for code output	✓	✓		The editor's standards-compliant markup meets or exceeds accessibility requirements: Section 508 (USA), CLF (Canada), etc.
Distinguishes between data and layout tables	✓	✓		Maintains the important distinction between <a href="#">data and layout tables</a> , which are processed differently by assistive technologies such as screen readers.
Distinguishes between decorative and informative images	✓	✓		XStandard makes it easy to ensure that informative images are used in a <a href="#">semantically meaningful</a> way.
Offers a rich API for extending the editor's functionality	✗	✓		<a href="#">Extend the functionality of XStandard</a> through custom programming to meet the unique needs of your content management system. Hook into XStandard events to launch your own dialog boxes and to programmatically insert markup into the editor.
Includes a unique "Screen Reader Preview"	✓	✓		The accessibility button on XStandard's toolbar opens the <a href="#">Screen Reader Preview</a> that helps authors further optimize content for accessibility.
Collapsible/expandable editor window via toolbar button	✓	✓		Collapsing the editor permits a more economic use of screen real estate, while permitting authors to expand the editor when in use. The editor can be expanded to a full-screen view, or customized to snap open to any size.

## Notes

 Supports most CSS 2.1 selectors 

### Universal selector

Matches the name of any element. For example:

```
* {margin:0;padding:0}
```

### Type selector

Matches the name of a given element. For example:

```
h1 {color:red}
```

### Descendant selector

Matches an element that is the descendant of another element. For example:

```
h1 em {color:red}
```

### Child selector

Matches an element that is the direct child of another element. For example:

```
p > q > strong {color:red}
```

### Adjacent sibling selector

Matches an element, given an element that immediately precedes it. For example:

```
h1 + p {margin-left:40px}
```

### Attribute selector

Matches elements that have certain attributes. For example:

```
a[title] {color:red}
```

### Class selector

Matches elements given a class value. For example:

```
em.important {color:red}
```

### ID selector

Matches an element with a given ID. For example:

```
#chapter1 {color:red}
```

## ❏ XStandard makes applying the correct CSS quick and easy 📈

Authors apply CSS by choosing formatting options from the editor's drop-down Styles menu. Each style in the menu generates markup that references an appropriate CSS. Since developers can attribute friendly, meaningful names to each style (for example, "Chapter Heading" or "Sale Price"), authors find it easy to recognize and apply the right formatting to the right content. Styles in the menu can also be grouped together for greater convenience, and the choice of styles will typically reflect the type of document being edited. For authors, these features make applying CSS an intuitive and comfortable experience that results in high levels of compliance with presentation standards.

## ❏ The advantages of using Web Services 📈

1. Web Services can be easily customized to meet the unique needs of different content management systems, whereas FTP lacks the programmatic hooks necessary for processes that require tighter integration. For example, requiring uploaded files to be stored directory in a database, pushed through a workflow process, or modified in some way before being saved to the server.
2. Today's applications also require more metadata than FTP is able to provide. For example, when uploading a file, Web Services can also supply the CMS with information such as the name of the document the file is associated with, the Session ID, which user uploaded the file, etc.
3. Some corporate networks also do not permit FTP, so Web Services that run over HTTP are an ideal solution.

## ❏ A configurable and customizable toolbar 📈

Both versions of XStandard permit toolbar icons to be hidden, displayed or moved to different positions on the editor's toolbar. XStandard Pro also has a customizable toolbar that allows developers to change the appearance of toolbar icons to match the look-and-feel of applications XStandard runs in, or to move frequently used styles off the drop-down Styles menu and onto the toolbar. Buttons can also be programmed to insert code snippets into the editor, or to extend the editor's functionality in other ways.

## ❏ Imports third-party data 📈

XStandard's "Directory" service is a timesaving feature that communicates with third-party applications, such as CMS. It allows users to import external data from those applications directly into the editor. Content imported in this way might include staff listings, product numbers and descriptions, or indeed any type of centrally stored information in any structure (tables, email addresses, etc.)

## ❏ Supports drag & drop of images directly into the editor, and file browsing 📈

Images can be dragged directly into XStandard from the desktop. Restrictions can be set on file size or type, and the dimensions of uploaded images are automatically calculated. XStandard ensures that images are used correctly by requiring images to be identified as "decorative" or "informative", and requiring `alt` text for the latter.

## ❏ Inserts custom tags that add semantic meaning to text and objects 📈

Custom tags allow business users to attach semantic meaning to elements of content during the authoring process. Custom tags can subsequently be used for indexing data, or for optimizing search results in enterprise-level search engines. They can also serve as placeholders for dynamic content. In the example below, a custom tag acts as a placeholder for the latest stock price:

```
1. The current stock price is $<stock exchange="NASDAQ">INTL</stock>.
```

At run time this gives the result:

```
1. The current stock price is $31.49.
```

## ❏ Supports the distinction between data and layout tables 📈

Data tables such as the one below contain data that can only be understood in relation to cell and column headers. If the association between the table cells and the headers is not made clear, non-visual browsers will read the data in linear

fashion. The result will be meaningless and [sound like this](#). By contrast, XStandard makes it easy to create tables that use `<th>` to identify the column and row headers that cell contents refer to. XStandard also requires authors to submit summaries for data tables, allowing non-visual browsers to describe tables fully. [Listen](#) to how a screen reader might process the same data table properly marked up using XStandard.

Cups of coffee consumed by each person

Name	Cups	Type	Sugar
Wendy	10	Regular	yes
Jim	15	Decaf	no

### Distinguishes between decorative and informative images

XStandard ensures that images are used correctly by prompting authors to identify images as decorative or informative, when images are uploaded through the editor or referenced in a remote library.

Decorative images are used for visual effect or as design elements (spacers or graphical bullets). Since they are not used to convey meaningful information, XStandard makes decorative images invisible to non-visual browsers by giving them an empty `alt` text, and by not requiring a `title` or `longdesc`.

By contrast, informative images such as photographs, diagrams and navigational aids do convey important meaning. XStandard therefore requires `alt` text for informative images and encourages users to also contribute both a `title` and `longdesc`. In addition, XStandard reinforces the distinction between `alt` text and `title` by asking for both. (`title` is properly used for tool tips, not `alt`)

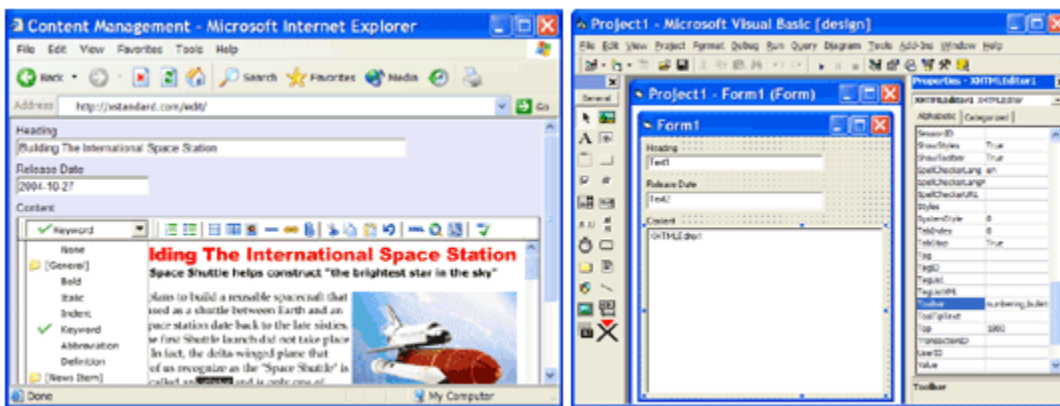
### Includes a unique "Screen Reader Preview"

Markup generated by XStandard is guaranteed to be accessibility-ready, but XStandard's Screen Reader Preview offers authors an additional opportunity to optimize content for accessibility. It does this by displaying content managed through XStandard as it is "read" by screen readers. This means content is laid out in linear fashion, together with information that the author would normally not see (alternate text, table summaries, tool tips, etc.). Previewing content in this fashion prompts authors to make necessary changes prior to publishing.

The Screen Reader Preview also issues alert messages when code entered manually through View Source contains semantically questionable markup. For example it discourages the use of the `<b>` tag, which has visual significance for sighted readers but carries no semantic significance for users of assistive technologies such as screen readers. In this case users would be encouraged to use `<strong>` which does have semantic meaning for screen readers. Other alert messages include warnings that alt text or table summaries are missing. This information must be submitted before content can be saved.

## Architecture

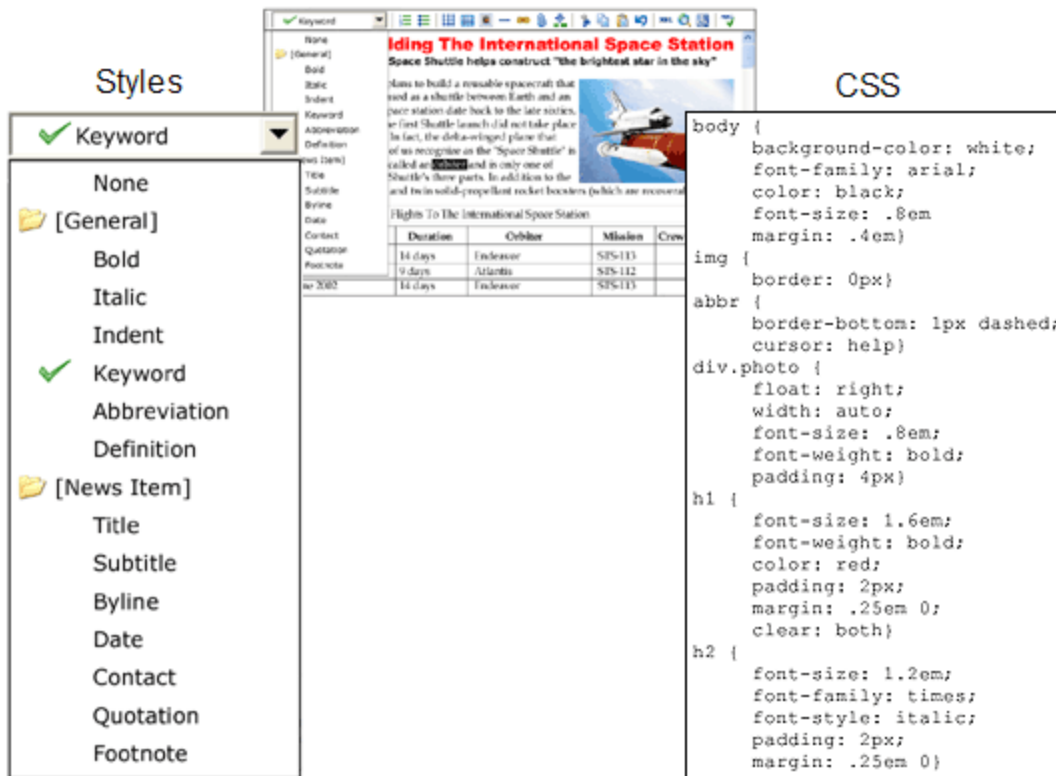
XStandard is written in C++ with a Firefox/Safari/Opera plug-in API and an ActiveX wrapper for IE. It runs natively in IE 5+, Firefox 1+, Safari 1.3+, Opera 9+ and many desktop development environments such as Visual Basic, Visual C++, Delphi, FoxPro, Access, Visual Studio.NET, etc.



## Separating Content From Presentation

XStandard generates clean, accessible, standards-compliant markup that separates content from presentation. To do this, XStandard avoids deprecated constructs like the `<FONT>` element and the `style` attribute that are typically created by font-selectors and color-pickers. Instead, as seen in the screen shot below, XStandard uses its Styles menu to create

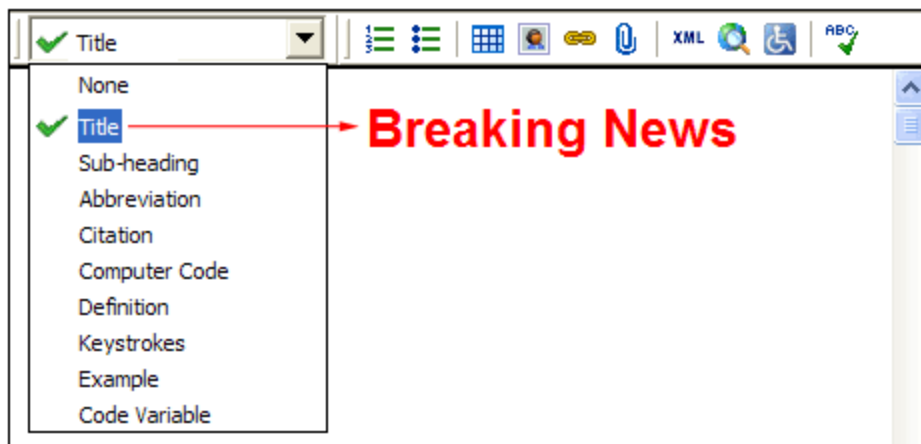
markup that is then formatted by CSS. The Styles menu (which uses convenient friendly names) makes applying the right formatting to the right content easy, and helps developers maintain presentation standards.



## Styles And CSS

Styles are instructions for creating markup. CSS formatting rules are applied to markup in order to create the desired presentation style.

Behind each user-friendly Style name are instructions for creating markup. For example, the style "Title" may create markup that looks like this `<h1 class="title">Breaking News</h1>`. CSS can then be used to format this markup. For example, `h1.title {color: red}`. The screen shot below illustrates this approach.



[Styles are customizable](#) and can be given any user-friendly name and can be instructed to create any element (with any number of attributes). [CSS are easy to write](#) and are a standards-compliant way of formatting Web content.

## Web Services

Web Services are applications that run on the server and communicate with other computers using a dialect of XML called SOAP (Simple Object Access Protocol). Typically, business users do not interact directly with Web Services. Instead, they interact with user-friendly programs which themselves communicate with Web Services. XStandard uses Web Services to upload files from local computers to the server, to build image, attachment and code



snippet libraries, for spell checking and for communicating with third-party applications (such as your CMS). See the [Web Services](#) section for more information.

## Installation

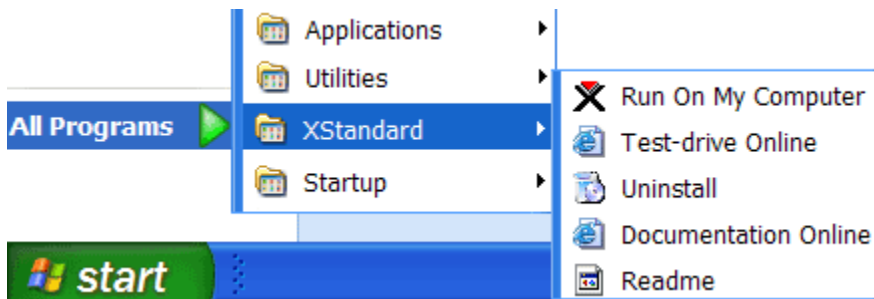
- [Windows](#)
  - [Install](#)
  - [Uninstall](#)
- [OS X 10.9 and earlier](#)
  - [Install](#)
  - [Uninstall](#)

## Windows

### Install

Download the installation program (x-lite.exe or x-pro.exe) and double-click on it to begin installing XStandard. The install wizard will guide you through the install process.

To test-drive XStandard, from the Start menu select "All Programs > XStandard > Test-drive Online" as shown in the screen shot below.



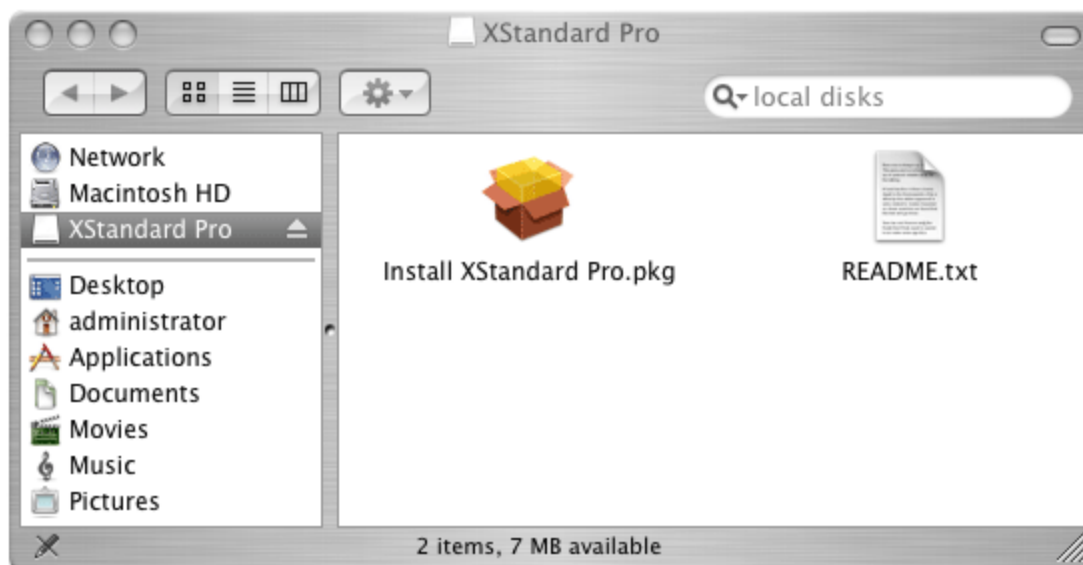
### Uninstall

To uninstall XStandard, from the Start menu select "All Programs > XStandard > Uninstall".

## OS X 10.9 and earlier

### Install

Download the installation program (x-lite.dmg or x-pro.dmg) and double click on it to begin installing XStandard. OS X will mount a drive called "XStandard Lite" or "XStandard Pro" and open it as shown in the screen shot below.



Double click on "Install XStandard Lite.pkg" or "Install XStandard Pro.pkg" and an install wizard will guide you through the install process. After the install, you can unmount the drive by dragging it to the Trash bin.

To test-drive XStandard, go to `/Applications/XStandard/test-drive.htm`

## Uninstall

To uninstall XStandard, delete the following:

- `/Applications/XStandard`
- `/Library/Internet Plug-Ins/XStandard.plugin`

## API Reference

XStandard is added to a Web page using an `<object>` tag. Inside the `<object>` tag, the `type` attribute is used to identify the type of plug-in to display, while `<param>` tags are used to customize the editor's functionality. For example:

```
<object type="application/x-xstandard" id="editor1" width="100%" height="400">
<param name="Value" value="Hello World!" />
</object>
```

Below is a list of properties (`<param>` tag names), methods, events and their descriptions.

- [Basic Settings](#)
- [Web Services Settings](#)
- [Customization Settings](#)
- [Authoring Techniques Settings](#)
- [Integration Settings](#)
- [Hooks & Extensions Settings](#)
- [Network Settings](#)
- [Miscellaneous Settings](#)

## Basic Settings

### Property *Value*

XHTML data that the user (business author) has entered.

### Property *CSS*

Absolute URL or file path to a CSS file or CSS data. If absent, a built-in CSS file will be used. This file contains CSS formatting rules and can be the same CSS file used on your Web site.

### Property *Styles*

Absolute URL or file path to an XML styles file or XML data as a string. If absent, a built-in styles file will be used. This file is used to populate the drop-down Styles Menu. Each style contains instructions for generating markup.

### Property *Base*

URL used to resolve relative URLs for images defined in markup.

### Property *License*

Absolute URL or file path to a license file or license data as a string. If absent, the editor will run under the XStandard Lite freeware license.

## Web Services Settings

### Property *SpellCheckerURL*

(Available in XStandard Pro)

Absolute URL to a Spell Web Service. For testing purposes, the following URL is available: `http://soap.xstandard.com/spellchecker.aspx`

### Property *SpellCheckerLangFilter*

(Available in XStandard Pro)

A comma-delimited list of dictionaries that are a sub-set of the dictionaries available for the Spell Web Service. In other words, if the Web Service supports 10 dictionaries but you want XStandard to use only 2, you would specify the two dictionaries in this parameter. For example: `en-ca, fr`

### Property *SpellCheckerLang*

(Available in XStandard Pro)

A language code to indicate a default dictionary language for spell checking. The default dictionary must be defined in the `spellchecker.config` file for the SpellChecker Web Service. Example: `en-us`

### Property *DirectoryURL*

(Available in XStandard Pro)

Absolute URL to a Directory Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available: `http://soap.xstandard.com/directory.aspx`

### Property *ImageLibraryURL*

(Available in XStandard Pro)

Absolute URL to an Image Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available: `http://soap.xstandard.com/imagelibrary.aspx`

### Property *AttachmentLibraryURL*

(Available in XStandard Pro)

Absolute URL to an Attachment Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available:

`http://soap.xstandard.com/attachmentlibrary.aspx`

### Property *LinkLibraryURL*

(Available in XStandard Pro)

Absolute URL to a Link Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available:

`http://soap.xstandard.com/linklibrary.aspx`

### Property *SubdocumentURL*

(Available in XStandard Pro)

Absolute URL to a Subdocument Web Service. For testing purposes, the following URL is available:

`http://soap.xstandard.com/subdocument.aspx`

## Customization Settings

These settings are used to configure the localization, functionality and look & feel of the editor.

### Property *Lang*

A two-letter language code (should be lowercase) that is used for localization. XStandard ships with 19 localizations

- `en` (English), `de` (German), `fr` (French), `nl` (Dutch), `es` (Spanish), `it` (Italian), `cn` (Simplified Chinese), `id` (Indonesian), `ja` (Javanese), `el` (Greek), `ru` (Russian), `uk` (Ukrainian), `he` (Hebrew), `sr` (Serbian), `sh` (S erbo-Croatian), `cs` (Czech), `da` (Danish), `fi` (Finnish) and `sv` (Swedish). The default value is `en`. See

the [Localization](#) section of this document for how to create your own localizations.

### Property *Dir*

Sets the default text direction for the editor. Possible values are: `ltr` and `rtl`.

### Property *EditorCSS*

Absolute URL or file path to a CSS file or CSS data. This file contains CSS rules that are only used by the editor in WYSIWYG mode. This property is useful for locking and markers. The CSS rules can be set at run-time like this:

```
XHTMLEditor1.EditorCSS = "h1 {-xs-lock: yes; color:red}";
```

### Property *EnablePasteMarkup*

(Available in XStandard Pro)

If set to `yes` (or `True`), the editor will attempt to retain structural elements (lists, tables, hyperlinks, images, headings, etc.) when pasting from Microsoft Word or from Web pages. Semantically meaningless markup such as `<font>` will be stripped out. The default value is `no` (or `False`).

### Property *EnableTimestamp*

(Available in XStandard Pro)

This setting is used to enable or disable the timestamp comment added to markup generated by the editor. The default value is `yes`. The timestamp looks like:

```
<!-- Generated by XStandard version 2.0.0.0 on 2007-06-15T14:16:20.967 -->
```

## Property *Localization*

Absolute URL or file path to localization.xml file or XML data as a string. If absent, US English localization will be used.

## Property *PreviewXSLT*

Absolute URL or file path to preview.xsl file or XSLT data as a string. If absent, a built-in preview file will be used. This file is used to render the Browser Preview feature.

## Property *ScreenReaderXSLT*

Absolute URL or file path to screenreader.xsl file or XSLT data as a string. If absent, a built-in screen reader preview file will be used. This file is used to render the Screen Reader Preview feature.

## Property *Buttons*

(Available in XStandard Pro)

Absolute URL or file path to buttons.xml file or XML data as a string. If absent, a built-in buttons file will be used. This file is used to change icons and define buttons for the toolbar. The property `ToolbarWysiwyg` is used to specify which of the buttons defined in this file will appear on the toolbar.

## Property *Icons*

(Available in XStandard Pro)

Absolute URL or file path to icons.xml file or XML data as a string. If absent, a built-in icons file will be used. This file is used to change icons used in the editor.

## Property *Placeholders*

(Available in XStandard Pro)

Absolute URL or file path to placeholders.xml file or XML data as a string. If absent, a built-in placeholders file will be used. This file is used to change the icons used for placeholders (custom tags). See the [Placeholders](#) section of this document for more details.

## Property *BackgroundColor*

This is a Web-named color or a HEX color value for the background of solid areas in the editor, such as the toolbar. For example: `yellow` or `#ffff00`. This feature is not available for Mozilla/Firefox on Windows XP. Note that the background color used in the actual authoring area of the editor is controlled by CSS.

## Property *BorderColor*

This is a Web-named color or a HEX color value for the editor's border.

## Property *ToolbarWysiwyg*

This parameter customizes the toolbar in WYSIWYG mode. This parameter accepts a comma-delimited list of button IDs. Buttons IDs are defined in buttons.xml file. A list of default buttons IDs is available in the [Interface](#) section of this documentation. Consecutive commas generate a delimiter (vertical bar) between buttons. For example:

```
ordered-list, unordered-list, definition-list,, draw-layout-table, draw-data-table,
image, separator, hyperlink, attachment, directory, spellchecker,, wysiwyg, source,
preview, screen-reader, help
```

To force the toolbar to wrap, use `;` at the wrapping point.

## Property *ToolbarSource*

This parameter customizes the toolbar in View Source mode. This parameter accepts a comma-delimited list of button IDs. Buttons IDs are defined in buttons.xml file. A list of default buttons IDs is available in the [Interface](#) section of this documentation. Consecutive commas generate a delimiter (vertical bar) between buttons. For example:

```
indent, whitespace, word-wrap, dim-tags, validate,, wysiwyg, source, preview, screen-
reader
```

## Property *ToolbarPreview*

This parameter customizes the toolbar in Preview mode. This parameter accepts a comma-delimited list of button IDs. Buttons IDs are defined in buttons.xml file. A list of default buttons IDs is available in the [Interface](#) section of this documentation. Consecutive commas generate a delimiter (vertical bar) between buttons. For example:

```
wysiwyg, source, preview, screen-reader
```

## Property *ToolbarScreenReader*

This parameter customizes the toolbar in Screen Reader Preview mode. This parameter accepts a comma-delimited list of button IDs. Buttons IDs are defined in buttons.xml file. A list of default buttons IDs is available in the [Interface](#) section of this documentation. Consecutive commas generate a delimiter (vertical bar) between buttons. For example:

```
wysiwyg, source, preview, screen-reader
```

## Property *ToolbarEffect*

This parameter is used to give a decorative effect to the background of the toolbar. Currently on value of "linear-gradient" is supported.

## Property *ShowStyles*

(Available in XStandard Pro)

This parameter can be used to hide the drop-down Styles Menu found on the toolbar. Possible values are: `yes` and `no`.

## Property *ShowToolbar*

(Available in XStandard Pro)

This parameter can be used to hide toolbar buttons. Possible values are: `yes` and `no`.

## Property *ExpandWidth*

(Available on Windows)

This parameter sets the width in pixels or as percent of screen width for the editor's expanded window.

## Property *ExpandHeight*

(Available on Windows)

This parameter sets the height in pixels or as percent of screen height for the editor's expanded window.

## Property *ExpandToolbarWysiwyg*

(Available on Windows)

This parameter customizes the toolbar in the editor's expanded window in WYSIWYG mode.

See `ToolbarWysiwyg` parameter for setting values.

## Property *ExpandToolbarSource*

(Available on Windows)

This parameter customizes the toolbar in the editor's expanded window in View Source mode.

See `ToolbarSource` parameter for setting values.

## Property *ExpandToolbarPreview*

(Available on Windows)

This parameter customizes the toolbar in the editor's expanded window in Preview mode.

See `ToolbarPreview` parameter for setting values.

## Property *ExpandToolbarScreenReader*

(Available on Windows)

This parameter customizes the toolbar in the editor's expanded window in Screen Reader Preview mode.

See `ToolbarScreenReader` parameter for setting values.

## Property *ExpandShowStyles*

(Available in XStandard Pro on Windows)

This parameter can be used to hide the drop-down Styles Menu found on the toolbar in the editor's expanded window.

Possible values are: `yes` and `no`.

## Property *ExpandShowToolbar*

(Available in XStandard Pro on Windows)

This parameter can be used to hide toolbar buttons in the editor's expanded window. Possible values are: `yes` and `no`.

## Property *CustomInlineElements*

A comma delimited list of custom elements the editor will treat as inline elements. For example: `price, stock, temperature`. Custom inline elements are treated like a `<span>`. Inline elements cannot contain block elements. Inline elements must be contained within a block element. For example:

```
<p>Today's temperature is <temperature />.</p>
```

## Property *CustomBlockElements*

A comma delimited list of custom elements the editor will treat as block elements. For example: `include, rss`. Custom block elements are treated like a `<div>` and therefore cannot be contained by `<p>`. For example:

```
<p>Some text</p>
<include doc="123" />
<p>Some more text</p>
```

## Property *CustomEmptyElements*

A comma delimited list of custom elements the editor will treat as empty elements. For example: `include`. Empty elements cannot contain content and are written like this: `<include doc="123" />`.

## Property *Rel*

A space delimited list of values used to populate the "Relationship" field in the Hyperlink properties dialog box. If this property contains data, it will display in the simplified interface (when `Options` 512 is set). Please note, this functionality will continue to be supported in future releases but the implementation (API) may change.

### Property Rev

A space delimited list of values used to populate the "Reverse relationship" field in the Hyperlink properties dialog box. If this property contains data, it will display in the simplified interface (when `Options` 512 is set). Please note, this functionality will continue to be supported in future releases but the implementation (API) may change.

### Property Options

Used to enable/disable features in the editor. This value is a bitmask (a sum of values associated with different features). The following table describes each setting.

Value	Description
1	Display hard returns.
2	Automatically fix errors when "dirty" code is loaded into the editor.
4	Hide most items from the context menus (pop-up menus when the right mouse button is clicked).
8	Wrap text in View Source.
16	Hide line numbers in View Source.
32	Disable object resizing by dragging (using image "handles" for example).
64	Treat uploading images as decorative.
128	Do not switch to Images As Text mode during find/replace or spell checking.
256	Convert extra spaces into hard-spaces.
512	Hide advanced editing features (Advanced and Custom tabs as well as certain fields in Properties dialog boxes).
1024	Use SOAP 1.2
2048	Treat <code>&lt;div&gt;</code> as as structure element. By default, <code>&lt;div&gt;</code> is used as a grouping element.
4096	Disable Directory button when content is selected.
8192	Disable "Fix" option when invalid markup is entered into View Source and user switches to WYSIWYG view.
16384	Automatically remove undefined custom elements. To define custom elements, use <code>CustomInlineElements</code> and <code>CustomBlockElements</code> properties.
32768	Paste images as alternate text when file upload is unavailable/disabled.
65536	Strict mode when pasting from applications like Word. Removes attributes like <code>id</code> , <code>align</code> and <code>valign</code> .
131072	Display the "Class" field on the General tab of the Properties dialog box in the simplified interface (when <code>Options</code> 512 is set). The only exception to this is in the Image properties dialog box when there is a value in <code>ClassImageFloatLeft</code> / <code>ClassImageFloatRight</code> properties.
262144	Display the "ID" field on the General tab of the Properties dialog box in the simplified interface (when <code>Options</code> 512 is set).

For example, a value of `24` (8 + 16) would both wrap text and hide line numbers in View Source.

## Authoring Techniques Settings

### Property `ClassImageFloatLeft`

A CSS class name used to align images to the left.

## Property *ClassImageFloatRight*

A CSS class name used to align images to the right.

## Property *ScriptNewWindow*

JavaScript used to open a new window. Default value is:

```
window.open(this.href);return false;
```

# Integration Settings

These settings are useful when integrating the editor into content management solutions.

## Property *EscapeUnicode*

If set to `yes` (or `True`), characters outside the ANSI limits will be escaped into a numeric representation. For example, `&#1084;&#1080;&#1088;`. This feature is useful when you need to work in a truly multilingual environment but some of the components of your content management system do not yet support Unicode.

## Property *DocumentID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Document-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_DOCUMENT_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_DOCUMENT_ID"]`.

## Property *UserID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-User-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_USER_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_USER_ID"]`.

## Property *SessionID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Session-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_SESSION_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_SESSION_ID"]`.

## Property *TransactionID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Transaction-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_TRANSACTION_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_TRANSACTION_ID"]`.

## Property *ClientID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Client-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_CLIENT_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_CLIENT_ID"]`.

## Property *InstanceID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Instance-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_INSTANCE_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_INSTANCE_ID"]`.

## Property *TagID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Tag-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_TAG_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_TAG_ID"]`.

## Property *ZoneID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Zone-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_ZONE_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_ZONE_ID"]`.

### Property *ProjectID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Project-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_PROJECT_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_PROJECT_ID"]`.

### Property *AreaID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Area-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_AREA_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_AREA_ID"]`.

### Property *GroupID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Group-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_GROUP_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_GROUP_ID"]`.

### Property *ParentID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Parent-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_PARENT_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_PARENT_ID"]`.

### Property *ContainerID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Container-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_CONTAINER_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_CONTAINER_ID"]`.

### Property *ObjectID*

Used for integration with CM systems, the value of this property will be transmitted in all HTTP requests made by the editor in a header named: `X-Object-ID`. From ASP, read this value

as: `Request.ServerVariables("HTTP_X_OBJECT_ID").Item`. From PHP, read this value

as: `$_SERVER["HTTP_X_OBJECT_ID"]`.

### Property *Cookie*

This value is used to maintain a cookie-based session state between the editor and Web Services. Web Services have to be configured to use session state and the value for this property is typically set as follows:

ASP example: `<param name="Cookie"`

`value="<%=Server.HtmlEncode(Request.ServerVariables("HTTP_COOKIE").Item) %>" />`

PHP example: `<param name="Cookie" value="<?php echo`

`htmlspecialchars($_SERVER["HTTP_COOKIE"], ENT_COMPAT) ?>" />`

### Property *EnableCache*

Caches the editor's customization files when downloaded over a network. See the [Caching](#) section in this document for more information on configuring this feature.

### Property *HeartbeatURL*

(Available in XStandard Pro)

Absolute URL to where "Heartbeat" pulses are sent. See the [Heartbeat](#) section of this document for more information on configuring this feature.

### Property *HeartbeatInterval*

(Available in XStandard Pro)

Number of seconds between Heartbeat pulses. The default value and minimum value is `60`.

### Property *Namespaces*

Namespaces declaration. For example:

```
<param name="Namespaces" value="xmlns:a='http://apple-books' xmlns:b='http://big-books'" />
```

See the [Namespaces](#) section of this document for more information on configuring this feature.

## Hooks & Extensions Settings



These settings are used to extend the functionality of the editor through custom code.

## Property Mode

(Available in XStandard Pro)

This property can be used to set the default view of the editor, or to programmatically switch between views. Possible values are: `wysiwyg` or `source` or `preview` or `screen-reader`. The default value is `wysiwyg`.

## Event ModeChanged(sFrom, sTo)

(Available in XStandard Pro in non-browser applications.)

Fires when `Mode` changes. To capture this event in Web-based applications, use the following code:

```
<script type="text/javascript">
//
function xsModeChanged(id) {
alert('Editor: ' + id + '; function: xsModeChanged()');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 293 290 311" data-label="Section-Header"><h2>Event ContentChanged()</h2></div><div data-bbox="55 310 267 325" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 324 936 353" data-label="Text"><p>Fires once when content is changed for the first time. To capture this event in Web-based applications, use the following code:</p></div><div data-bbox="74 357 646 457" data-label="Text"><pre>&lt;script type="text/javascript"&gt;
//<![CDATA[
function xsContentChanged(id) {
alert('Editor: ' + id + '; function: xsContentChanged()');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 461 410 479" data-label="Section-Header"><h2>Event ButtonClicked(sButton, sState)</h2></div><div data-bbox="55 478 267 493" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 492 938 524" data-label="Text"><p>Occurs when a button (of type <code>&lt;button&gt;</code> in <code>buttons.xml</code> file) is pressed. Possible values for <code>sState</code> are: <code>on</code> or <code>off</code>. To capture this event in Web-based applications, use the following code:</p></div><div data-bbox="74 526 823 626" data-label="Text"><pre>&lt;script type="text/javascript"&gt;
//<![CDATA[
function xsButtonClicked(id, button, state) {
alert('Editor: ' + id + '; function: xsButtonClicked(); button: ' + button);
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 630 346 648" data-label="Section-Header"><h2>Event ContextMenuActivated()</h2></div><div data-bbox="55 647 267 662" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 661 912 677" data-label="Text"><p>Occurs when the context menu is activated. To capture this event in Web-based applications, use the following code:</p></div><div data-bbox="74 680 734 865" data-label="Text"><pre>&lt;script type="text/javascript"&gt;
//<![CDATA[
function xsContextMenuActivated(id) {
alert('Editor: ' + id + '; function: xsContextMenuActivated()');
document.getElementById(id).ClearContextMenu();
document.getElementById(id).AddToContextMenu('a', 'My item a', '');
document.getElementById(id).AddToContextMenu('b', 'My item b', '');
document.getElementById(id).AddToContextMenu('c', 'My item c', '');
document.getElementById(id).AddToContextMenu('d', 'My item d', '');
document.getElementById(id).AddToContextMenu('e', 'My item e', '');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 868 389 886" data-label="Section-Header"><h2>Event ContextMenuClicked(sMenu)</h2></div><div data-bbox="55 885 267 901" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 899 907 931" data-label="Text"><p>Occurs when a custom item in the context menu is clicked. To capture this event in Web-based applications, use the following code:</p></div><div data-bbox="74 933 383 948" data-label="Text"><pre>&lt;script type="text/javascript"&gt;</pre></div>
```

```
//<![CDATA[  
function xsContextMenuClicked(id, menu) {  
alert('Editor: ' + id + '; function: xsContextMenuClicked(); menu: ' + menu);  
}  
//]]>  
</script>
```

### **Sub AddToContextMenu(*sID* As String, *sName* As String, *sGroup* As String)**

(Available in XStandard Pro)

Add custom item to the context menu.

`sID` is the ID for the context menu item.

`sName` is the label for the context menu item.

`sGroup` is reserved for future use. This argument can be set to empty string.

### **Sub ClearContextMenu()**

(Available in XStandard Pro)

Remove custom items from the context menu.

### **Sub CallToolbarButton(*sID*)**

(Available in XStandard Pro)

Simulates a toolbar button click.

### **Sub ApplyStyleID(*sID*)**

(Available in XStandard Pro)

Add or remove a style referenced by an ID. The style can be defined in the drop-down Styles Menu found on the toolbar.

### **Sub ApplyStyleXML(*sStyle*)**

(Available in XStandard Pro)

Add or remove a style given the XML of the style.

### **Function CurrentStyles()**

(Available in XStandard Pro)

A comma-delimited list of style IDs for the current cursor position. For example: `1,2,3,4,5`. This list comes from the styles defined in the drop-down Styles Menu (not the toolbar). If `/style/id` is missing from the style's XML definition, then this style will be omitted.

### **Property SelectedText**

(Available in XStandard Pro)

Text of the current selection. If an empty element is selected (such as `<img>`) or if nothing at all is selected, an empty string is returned.

### **Property SelectedXML**

(Available in XStandard Pro)

Markup of the current selection. If nothing is selected, an empty string is returned.

### **Property TagList**

(Available in XStandard Pro)

A comma-delimited list of parent tags (including the current tag) relative to the current cursor position. For example: `body,p,strong`

### **Property Path**

(Available in XStandard Pro)

TagList as an XPath expression. For example: `/body/p/strong`

### **Property QPath**

(Available in XStandard Pro)

TagList as a qualified XPath expression. For example: `/body[1]/p[2]/strong[4]`

### **Property TagListXML**

(Available in XStandard Pro)

An XML document with a list of parent tags (including the current tag) relative to the current cursor position. For example:

```
<taglist>  
<elt>  
<name>body</name>  
<index>1</index>  
</elt>  
<elt>  
<name>p</name>  
<index>2</index>  
<attr>
```

```
<name>class</name>
<value>indent</value>
</attr>
</elt>
<elt>
<name>strong</name>
<index>4</index>
</elt>
</taglist>
```

### Event **TagListChanged()**

(Available in XStandard Pro)

Occurs when the taglist changes for the current cursor position. To capture this event in Web-based applications, use the following code:

```
<script type="text/javascript">
//
function xsTagListChanged(id) {
alert('Editor: ' + id + '; function: xsTagListChanged()');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 345 275 362" data-label="Section-Header"><h3>Sub <b>InsertXML(sValue)</b></h3></div><div data-bbox="55 362 267 377" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 377 383 392" data-label="Text"><p>Inserts markup at the current cursor position.</p></div><div data-bbox="55 392 275 408" data-label="Section-Header"><h3>Sub <b>InsertText(sValue)</b></h3></div><div data-bbox="55 408 267 423" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 423 356 438" data-label="Text"><p>Inserts text at the current cursor position.</p></div><div data-bbox="55 437 447 455" data-label="Section-Header"><h3>Sub <b>SetAttribute(sQPath, sName, sValue)</b></h3></div><div data-bbox="55 454 267 469" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 469 741 485" data-label="Text"><p>Inserts or updates an attribute for a given element identified by the qualified XPath expression.</p></div><div data-bbox="55 484 419 501" data-label="Section-Header"><h3>Sub <b>RemoveAttribute(sQPath, sName)</b></h3></div><div data-bbox="55 501 267 516" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 516 667 531" data-label="Text"><p>Deletes an attribute for a given element identified by the qualified XPath expression.</p></div><div data-bbox="55 530 273 548" data-label="Section-Header"><h3>Function <b>Fix(sMarkup)</b></h3></div><div data-bbox="55 548 267 562" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 562 948 592" data-label="Text"><p>This function takes "dirty" / legacy markup and attempts to clean it. Use this function when you need to clean HTML generated by other WYSIWYG editors, or when returning data from Internet Explorer's <code>innerHTML</code> property.</p></div><div data-bbox="55 592 222 610" data-label="Section-Header"><h3>Event <b>FocusSet()</b></h3></div><div data-bbox="55 609 340 625" data-label="Text"><p>Occurs when the editor receives focus.</p></div><div data-bbox="55 640 232 658" data-label="Section-Header"><h3>Event <b>FocusLost()</b></h3></div><div data-bbox="55 657 318 672" data-label="Text"><p>Occurs when the editor loses focus.</p></div><div data-bbox="55 687 411 705" data-label="Section-Header"><h3>Sub <b>CallProperties(sQPath As String)</b></h3></div><div data-bbox="55 704 267 719" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 719 612 734" data-label="Text"><p>Programmatically bring up the Properties dialog box for a given element. For</p></div><div data-bbox="55 733 454 750" data-label="Text"><p>example: <code>.CallProperties("/body[1]/h2[1]")</code></p></div><div data-bbox="55 750 225 768" data-label="Section-Header"><h3>Sub <b>ClearCache()</b></h3></div><div data-bbox="55 768 740 785" data-label="Text"><p>Programmatically removes <a href="#">cached configuration files</a>. This method is used in conjunction</p></div><div data-bbox="55 783 943 815" data-label="Text"><p>with <code>EnableCache</code> property. It is equivalent to manually selecting <code>Editor &gt; Clear private data</code> from the context menu.</p></div><div data-bbox="55 814 542 834" data-label="Section-Header"><h3>Function <b>GetAttributes(sQPath As String) As String</b></h3></div><div data-bbox="55 832 267 848" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 847 609 863" data-label="Text"><p>Returns an <code>XML</code> document containing the attributes for a given element. For</p></div><div data-bbox="55 862 574 880" data-label="Text"><p>example, <code>.GetAttributes("/body[1]/p[1]/a[1]")</code> may return:</p></div><div data-bbox="75 883 276 940" data-label="Text"><pre>&lt;attributes&gt;
&lt;attr&gt;
&lt;name&gt;class&lt;/name&gt;
&lt;value&gt;intro&lt;/value&gt;</pre></div>
```

```
</attr>
<attr>
<name>href</name>
<value>/news/</value>
</attr>
</attributes>
```

### **Sub RemoveAttribute(sQPath As String, sName As String)**

(Available in XStandard Pro)

Removes an attribute from a given element.

### **Sub SetAttribute(sQPath As String, sName As String, sValue As String)**

(Available in XStandard Pro)

Adds/updates an attribute for a given element.

### **Event DialogPropertiesActivated(sQPath As String, sElement As String, sAttributes As String, sMetadata As String)**

(Available in XStandard Pro)

Occurs when the Properties dialog box is requested. This event can be used to replace the editor's Properties dialog box with your own Properties dialog box. Below is a description of each argument:

`sQPath` is a qualified XPath expression of the current element being edited. This value can be blank if the element has not been inserted into markup yet.

`sElement` is the name of the element being inserted/edited.

`sAttributes` is an XML document describing the attributes for this element. For example:

```
<attributes>
<attr>
<name>href</name>
<value>/files/report.doc</value>
</attr>
<attr>
<name>title</name>
<value>Year 2009 Annual Report</value>
</attr>
</attributes>
```

`sMetadata` is reserved for future use.

To pass to the editor attributes from your own dialog box and to cancel the editor's dialog box, call `SetDialogProperties()` during this event.

To capture this event in Web-based applications, use the following code:

```
<script type="text/javascript">
//
function xsDialogPropertiesActivated(id, qpath, element, attributes, metadata) {
alert('Editor: ' + id + '; function: xsDialogPropertiesActivated()');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 702 929 732" data-label="Text"><p>Note, for Web-based applications, do not use floating divs as dialog boxes because they will render behind the editor in the browser's z-order.</p></div><div data-bbox="55 746 943 781" data-label="Section-Header"><h3><b>Sub SetDialogProperties(sAttributes As String, bCancelDialog As Boolean, bCancelOperation As Boolean)</b></h3></div><div data-bbox="55 781 266 797" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 796 901 812" data-label="Text"><p>Passes attribute values to the editor or the Properties dialog box during <code>DialogPropertiesActivated()</code> event.</p></div><div data-bbox="55 811 354 828" data-label="Text"><p>Below is a description of each argument:</p></div><div data-bbox="55 826 723 843" data-label="Text"><p><code>sAttributes</code> is an XML document describing the attributes for this element. For example:</p></div><div data-bbox="75 847 393 946" data-label="Text"><pre>&lt;attributes&gt;
&lt;attr&gt;
&lt;name&gt;href&lt;/name&gt;
&lt;value&gt;/files/report.doc&lt;/value&gt;
&lt;/attr&gt;
&lt;attr&gt;
&lt;name&gt;title&lt;/name&gt;</pre></div>
```

```
<value>Year 2009 Annual Report</value>
</attr>
</attributes>
```

`bCancelDialog` is a way for your code to disable the editor's Properties dialog box. If the value is `True`, the editor will not display the Properties dialog box.

`bCancelOperation` is a way for your code to cancel modifications to the markup. For example, if the user presses "Cancel" button in your dialog box, you should set this value to `True`.

### **Event Paste(*bMarkup, sData*)**

(Available in XStandard Pro)

This event fires when content is pasted into the editor. `bMarkup` is a boolean value indicating if the content to be pasted is plain text or markup. `sData` contains the content to be pasted. To modify the pasted content before it is inserted into the editor, call `SetPaste()` method.

To capture this event in Web-based applications, use the following code:

```
<script type="text/javascript">
//
function xsPaste(id, markup, data) {
alert('Editor: ' + id + '; function: xsPaste()');
}
//]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="55 375 551 392" data-label="Section-Header"><h3><b>Sub SetPaste(<i>bMarkup As Boolean, sData As String</i>)</b></h3></div><div data-bbox="55 392 267 406" data-label="Text"><p>(Available in XStandard Pro)</p></div><div data-bbox="55 405 556 421" data-label="Text"><p>This method modifies the pasted content during the <code>Paste()</code> event.</p></div><div data-bbox="55 423 267 445" data-label="Section-Header"><h2><b>Network Settings</b></h2></div><div data-bbox="55 444 268 462" data-label="Section-Header"><h3><b>Property ProxySetting</b></h3></div><div data-bbox="55 462 454 478" data-label="Text"><p>Specify if a proxy should be used. Possible values are:</p></div><div data-bbox="55 491 286 507" data-label="Section-Header"><h4><b>auto-detect</b> (default value)</h4></div><div data-bbox="55 508 680 524" data-label="Text"><p>On Windows, the editor will use the connection settings specified for Internet Explorer.</p></div><div data-bbox="55 523 121 538" data-label="Section-Header"><h4><b>direct</b></h4></div><div data-bbox="55 538 433 554" data-label="Text"><p>Connect directly to the network. Do not use proxies.</p></div><div data-bbox="55 554 121 568" data-label="Section-Header"><h4><b>manual</b></h4></div><div data-bbox="55 568 489 585" data-label="Text"><p>Use the proxy settings specified by the following properties:</p></div><div data-bbox="64 599 213 670" data-label="List-Group"><ul><li>• ProxyServer</li><li>• ProxyPort</li><li>• ProxyUser</li><li>• ProxyPassword</li></ul></div><div data-bbox="55 672 141 687" data-label="Section-Header"><h4><b>platform</b></h4></div><div data-bbox="55 687 938 718" data-label="Text"><p>On Windows, this setting instructs the editor to use HTTP built into Internet Explorer. With this setting, the editor may be able to auto-login to your proxy server.</p></div><div data-bbox="55 717 264 734" data-label="Section-Header"><h3><b>Property ProxyServer</b></h3></div><div data-bbox="55 733 336 750" data-label="Text"><p>Name or IP address of a proxy server.</p></div><div data-bbox="55 764 243 782" data-label="Section-Header"><h3><b>Property ProxyPort</b></h3></div><div data-bbox="55 781 267 797" data-label="Text"><p>Port number of proxy server.</p></div><div data-bbox="55 810 246 829" data-label="Section-Header"><h3><b>Property ProxyUser</b></h3></div><div data-bbox="55 827 250 844" data-label="Text"><p>User name used by proxy.</p></div><div data-bbox="55 856 295 876" data-label="Section-Header"><h3><b>Property ProxyPassword</b></h3></div><div data-bbox="55 875 243 891" data-label="Text"><p>Password used by proxy.</p></div><div data-bbox="55 903 341 928" data-label="Section-Header"><h2><b>Miscellaneous Settings</b></h2></div><div data-bbox="55 926 220 945" data-label="Section-Header"><h3><b>Property Version</b></h3></div>
```

(read-only)

The version of the editor currently in use. For example: `3.0.0.0`

### Property *LatestVersion*

This optional value is the latest available version of the editor. If this value is greater than the version of the editor installed, a message box informs the user that a newer version of the product is available. For example: `3.0.0.0`

### Property *IndentOutput*

If set to `yes` (or `True`), the output of the code will be indented. The default value is `no` (or `False`).

### Property *Debug*

If set to `yes` (or `True`), to debug HTTP related issue. HTTP calls will be logged and can be seen on the Debug tab of the About dialog box. To open the About dialog box, from the context menu select "Editor > About". To avoid unnecessary writes to the log file, generally keep this value set to `no` (False).

### Property *Data*

Same as the Value property. Use this property in Microsoft Access or other development environments where the `Value` property is not available.

## Web Integration

- [Step 1](#)
- [Step 2](#)
- [Step 3](#)
- [Step 4](#)
- [Step 5](#)
- [Examples](#)
- [Integration FAQs](#)

## Step 1 - Adding XStandard To The Web Interface

XStandard runs in a Web browser as a plug-in. Plug-ins are added to Web pages using the `<object>` tag. An example of the `<object>` tag is below:

```
<object type="application/x-xstandard" id="editor1" width="100%" height="400">
<param name="Value" value="Hello World!" />
</object>
```

The `type` attribute contains a name that instructs the browser to load XStandard. The `id` attribute contains the ID that you assign to this instance of the editor. If you have multiple editors on the page, you must give each of them a unique ID. The `width` and `height` attributes define the dimensions of the editor.

The editor is configured through `<param>` tags which are inside the `<object>` tag. In the above example, the text `Hello World!` will be loaded into the editor. For a complete list of `<param>` tags, see [API Reference](#).

## Step 2 - Getting Data From XStandard

Unlike form elements, plug-in controls don't send data back to the server. To get around this, Web developers use a common technique that is reliable and easy to implement. Before the page is submitted, copy the data from the editor into a hidden field using JavaScript. When the form is submitted, the server-side script (ASP, PHP, ColdFusion, etc.) will read the data from the hidden field.

This is an example of JavaScript copying data from XStandard into a hidden field:

```
<html>
<head>
<script type="text/javascript">
function myOnSubmitEventHandler() {
document.getElementById('editor1').EscapeUnicode = true;
document.getElementById('xhtml1').value = document.getElementById('editor1').value;
}
</script>
</head>
```

```

<body>
<form method="post" onsubmit="myOnSubmitEventHandler()" action="mypage.htm">
<p>
<object type="application/x-xstandard" id="editor1" width="100%" height="400">
<param name="Value" value="Hello World!" />
</object>
</p>
<p>
<input type="hidden" name="xhtml1" id="xhtml1" value="" />
<input type="submit" name="btnAction" value="Submit" />
</p>
</form>
</body>
</html>

```

This is an example of a server-side script (in this case an ASP example) reading data from the hidden field:

```

<%
strXHTML = Request.Form("xhtml1").Item
%>

```

## Step 3 - Loading Data Into XStandard

Your content management system will probably store content in a database. In order to load this content into XStandard, your server-side script (ASP, PHP, ColdFusion, etc.) will read content from the database and put it into the `<param>` tag called "Value". This is an ASP example:

```

<object type="application/x-xstandard" id="editor1" width="100%" height="400">
<param name="Value" value="<%=Server.HTMLEncode(strXHTML)%>" />
</object>

```

In this example, the variable `strXHTML` contains the content from the database. Note, the content needs to be HTML escaped before it is loaded into the `<param>` tag. In ASP, this is done using the `Server.HTMLEncode()` function. See [Examples](#) for how to do this in ASP.NET, PHP, ColdFusion and JavaServer Pages.

## Step 4 - Referencing Configuration Files Styles And CSS

The `<param>` with the name "Styles" points to the location of the XML document that contains the styles.

The `<param>` with the name "CSS" points to the location of a CSS document. Use an absolute URL when pointing to these files. For example:

```

<param name="Styles" value="http://myserver/styles.xml" />
<param name="CSS" value="http://myserver/format.css" />

```

## License

A license file transforms XStandard Lite into Pro version. The `<param>` with the name "License" points to the location of the license file. Use an absolute URL when pointing to the license file. For example:

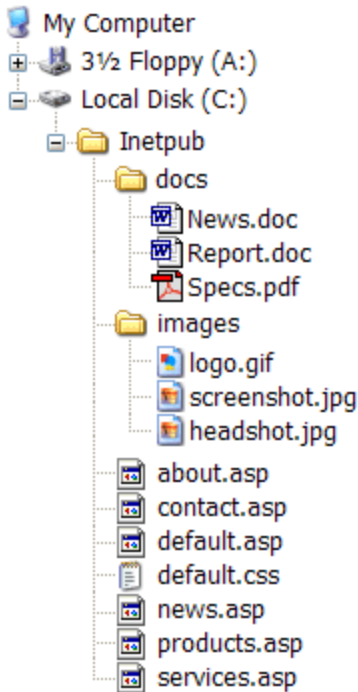
```

<param name="License" value="http://myserver/license.txt" />

```

## Step 5 - Uploading Images And Attachments To Libraries

Image and attachment libraries are folders on the Web server that contain files that are available to many Web pages. The screen shot below shows a typical website with both image and attachment libraries. Images are stored in a folder called "images" and attachments are stored in a folder called "docs".



To upload images and attachments to libraries, install the Image Library and Attachment Library Web Services on the Web server (see [Web Services](#) section) and set the following `<param>` tags to point to the Web Services.

```
<param name="ImageLibraryURL" value="http://myserver/images/imagelibrary.asp" />
<param name="AttachmentLibraryURL" value="http://myserver/docs/attachmentlibrary.asp" />
```

By default, the URLs for uploaded files are going to be absolute (the URLs will have the name of the server). To get XStandard to generate relative URLs, modify the variable `strBaseURL` in `imagelibrary.asp` and `attachmentlibrary.asp`.

For example:

```
strBaseURL = "images/"
```

In order for images with relative URLs to display correctly in the editor, specify the base URL in the following `<param>` tag:

```
<param name="Base" value="http://myserver" />
```

Free [technical support](#) is available to assist you in integrating XStandard into your solutions.

## Examples

Below are examples of how to integrate XStandard into different development environments.







## Web Integration FAQs

### General FAQs

#### *Can the editor be installed automatically in the browser?*

You can have the editor automatically installed in the browser the first time a Web page uses it.

- In IE, this is done via CAB file (see FAQs below).

#### *Can the editor be upgraded automatically in the browser?*

- In IE, this can be done via CAB file (see FAQs below).

#### *What is a CAB file?*

A cabinet is a single file created to hold a number of compressed files. During installation of a program, the compressed files in a cabinet are decompressed and copied to an appropriate directory.

#### *How is a CAB file used?*

When you are ready to go to production, you will want the editor to automatically install itself on the client machine when a Web page using the editor is opened for the first time. You can download the digitally signed CAB file for XStandard version 3.0.0.0 from:

<http://xstandard.com/download/XStandard.cab>

Put the CAB file on **your** Web server. Do not reference the CAB file from our Web site because we periodically update this file and the version number you use may not match the version of the CAB file on our Web site. Then reference it in the `<object>` tag like this:

```
<object type="application/x-xstandard" id="editor1" width="600" height="400"
codebase="http://yourserver/XStandard.cab#Version=3,0,0,0">
```

Don't forget the version number after the name of the CAB file in the `<object>` tag. The version number parts are separated by commas. The URL in the `codebase` param is case-sensitive, so make sure you use `XStandard.cab` instead of `xstandard.cab`.

As is common in IE, the *first* time a user opens a Web page using the editor, a security screen pops up asking users if they trust downloading the editor from Belus Technology. Note that this only happens once and, if you have a code signing digital certificate, you can sign the CAB files yourself, so that the message instead asks users if they trust software coming from `Your Company, Inc.`



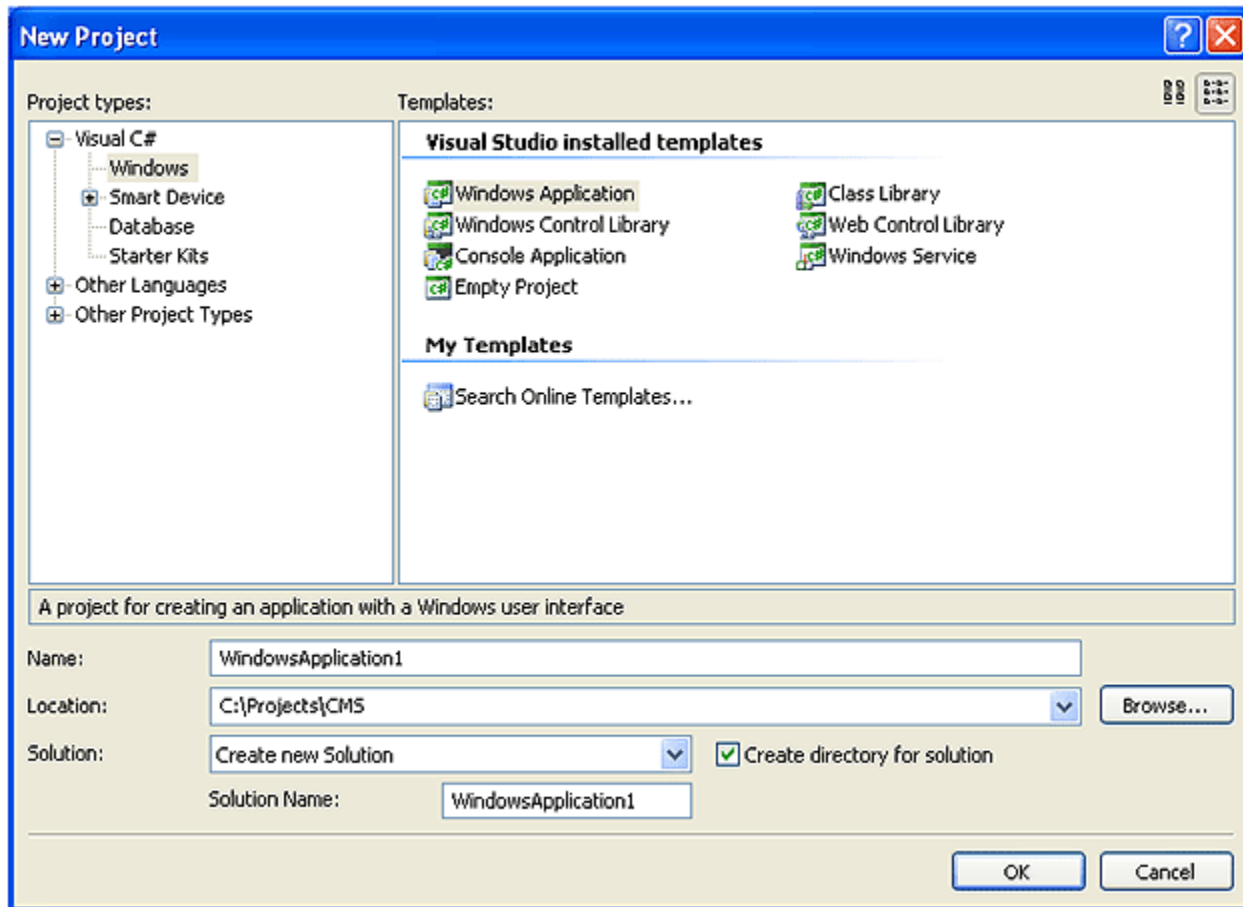
Microsoft IE 7 has a bug with auto-install. See [Knowledge Base](#) for a workaround.

## App Integration

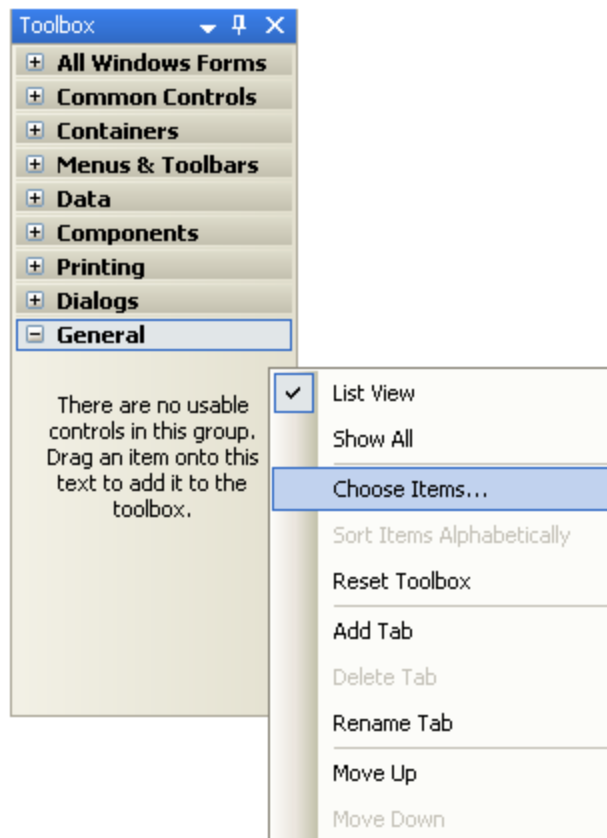
- [Visual Studio](#)
- [Access](#)
- [Visual Basic 6](#)
- [Visual C++ 6](#)
- [Delphi 7](#)
- [Visual FoxPro 9](#)

## Visual Studio .NET

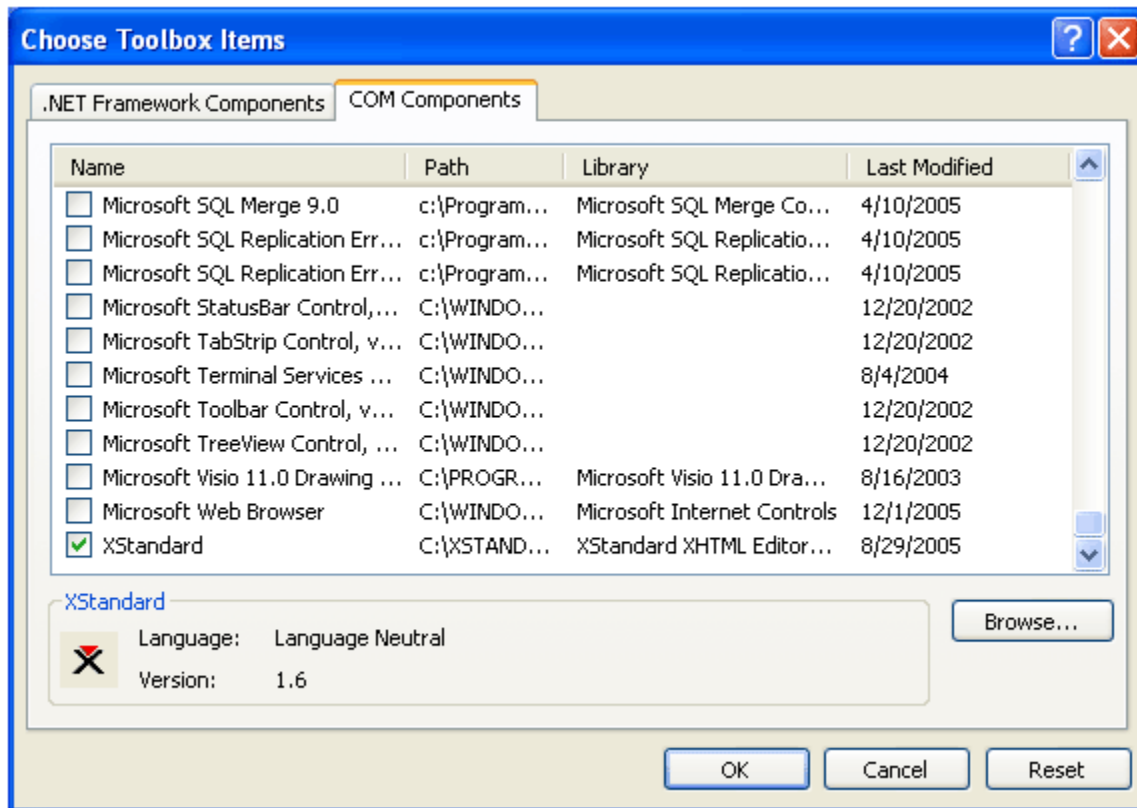
1. Start Visual Studio .NET and create a "Windows Application" as shown in the screen shot below.



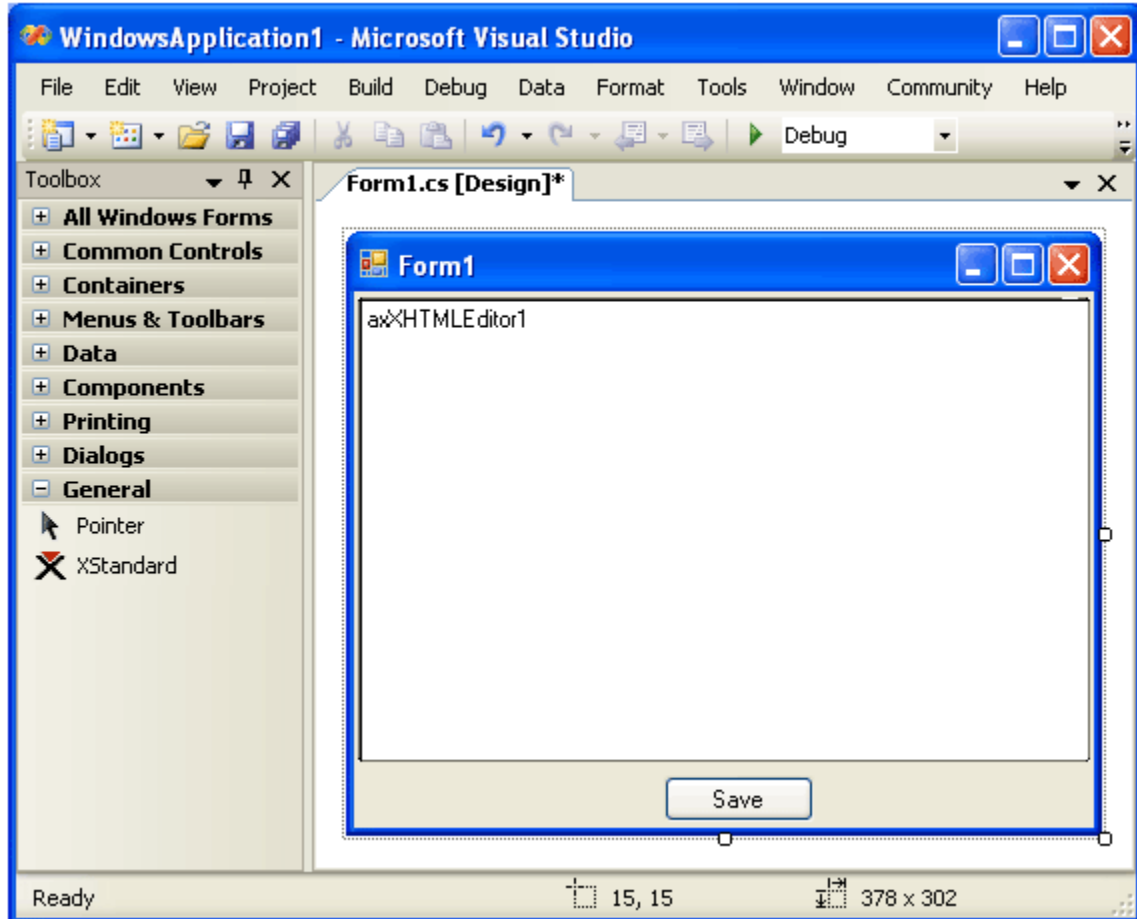
2. Open the "Toolbox" and right-click over the "General" selector. Select `Choose Items...` as show in the screen shot below.



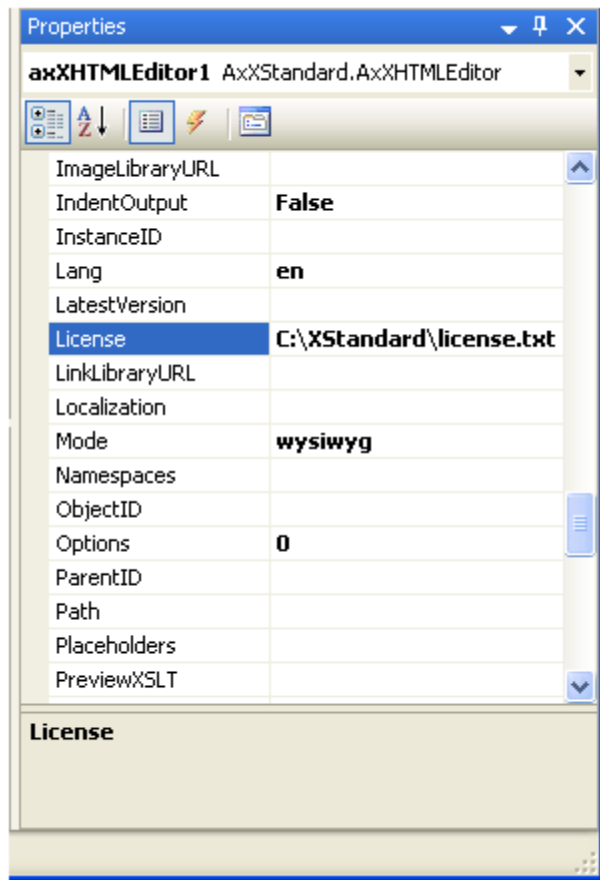
3. In the "Choose Toolbox Items" dialog box, select "COM Components" tab and check off "XStandard" at the bottom of the list as shown in the screen shot below.



4. XStandard will be added to the "Toolbox". Select it and add it to a form as shown in the screen shot below.



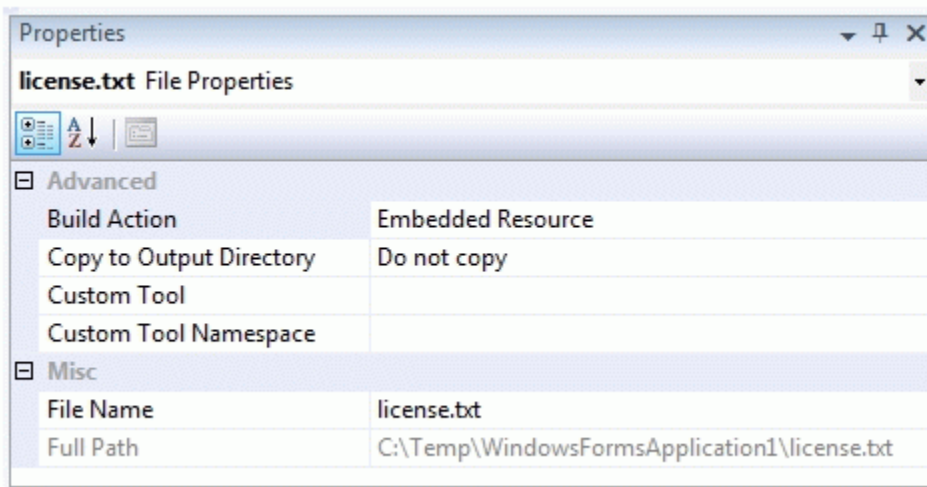
- Set the properties for the editor via the properties dialog box as shown in the screen shot below.



## Tips

### *Managing configuration files*

You can store the XML for the Styles drop-down list, CSS, XML file for the toolbar buttons and the license file as an Embedded Resource. You can then programmatically reference these files. For example, drag the license.txt file into your project. Click on the license.txt file in the Solution Explorer and in the Properties windows, set the Build Action to Embedded Resource as shown in the screen shot below.



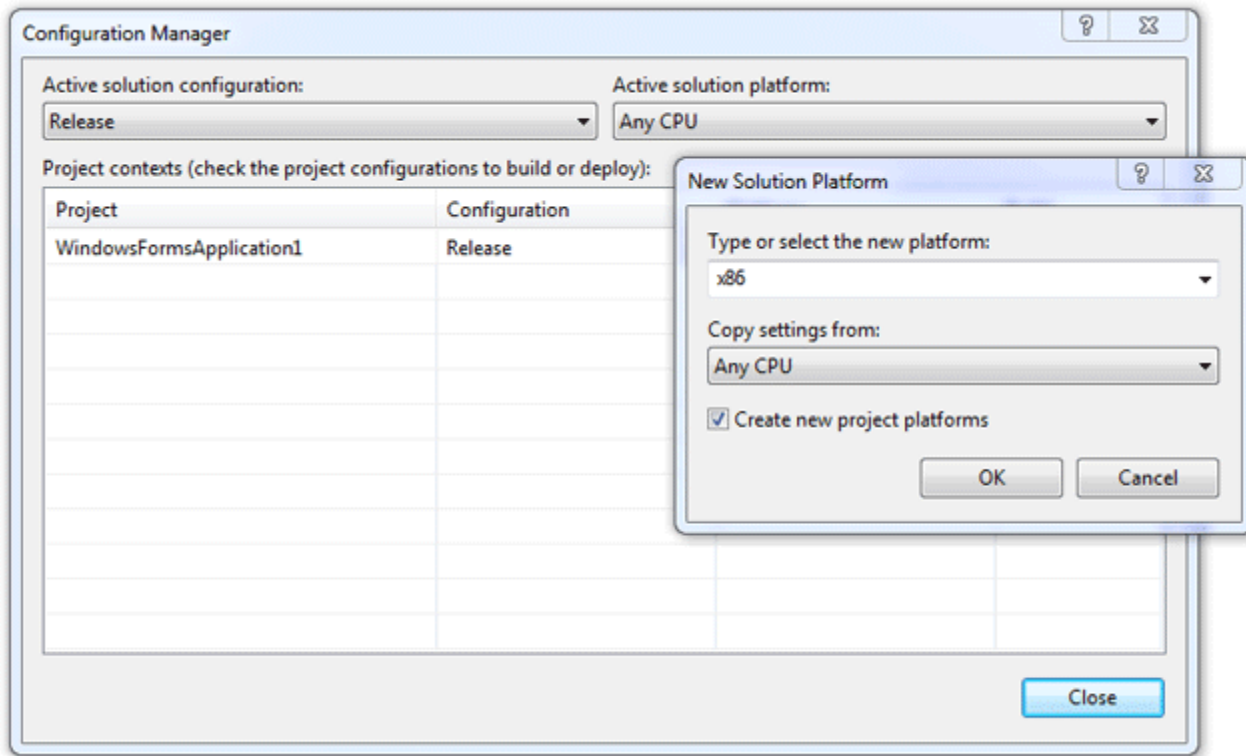
The text of the license file can then be passed to the editor's License property like this:

```
System.IO.Stream s;  
byte[] b;  
System.Text.StringBuilder code = new System.Text.StringBuilder();
```

```
s = System.Reflection.Assembly.GetExecutingAssembly().GetManifestResourceStream("WindowsFormsApplication1.license.txt");
b = new byte[System.Convert.ToInt32(s.Length)];
s.Read(b, 0, System.Convert.ToInt32(s.Length));
axXHTMLEditor1.License = System.Text.ASCIIEncoding.ASCII.GetString(b);
```

## Using XStandard on 64-bit OS

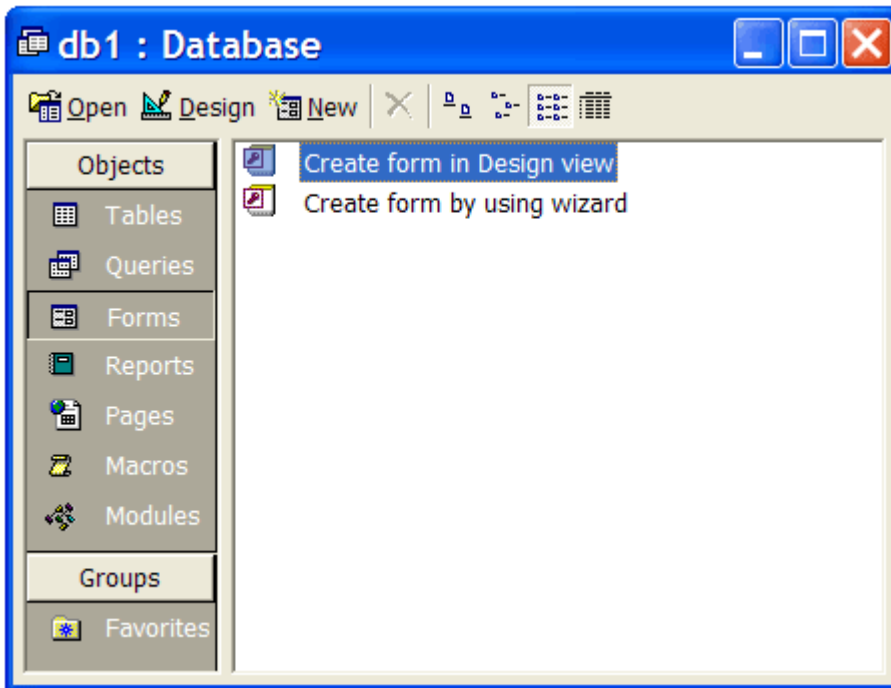
XStandard is a 32-bit component so you will need to compile your .NET WebForms application to x86 platform. From the Build menu, select "Configuration Manager...". In the "Active solution platform" field, select x86. If this value is not present, select <New...> and in the "New Solutions Platform" pop-up, select x86 in the drop-down list as shown in the screen shot below.



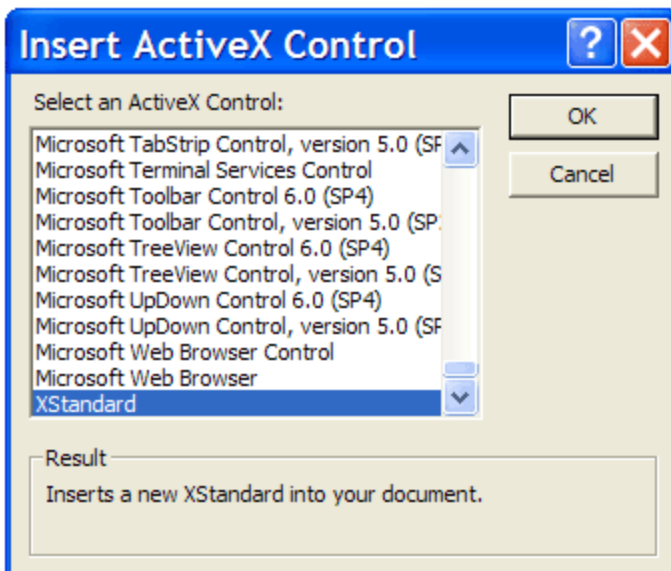
## Access 2000

1. Start Access and create a blank database.

2. Select "Forms" from the "Objects" list and double click on "Create form in Design view" as show in the screenshot below. This will create a new form.

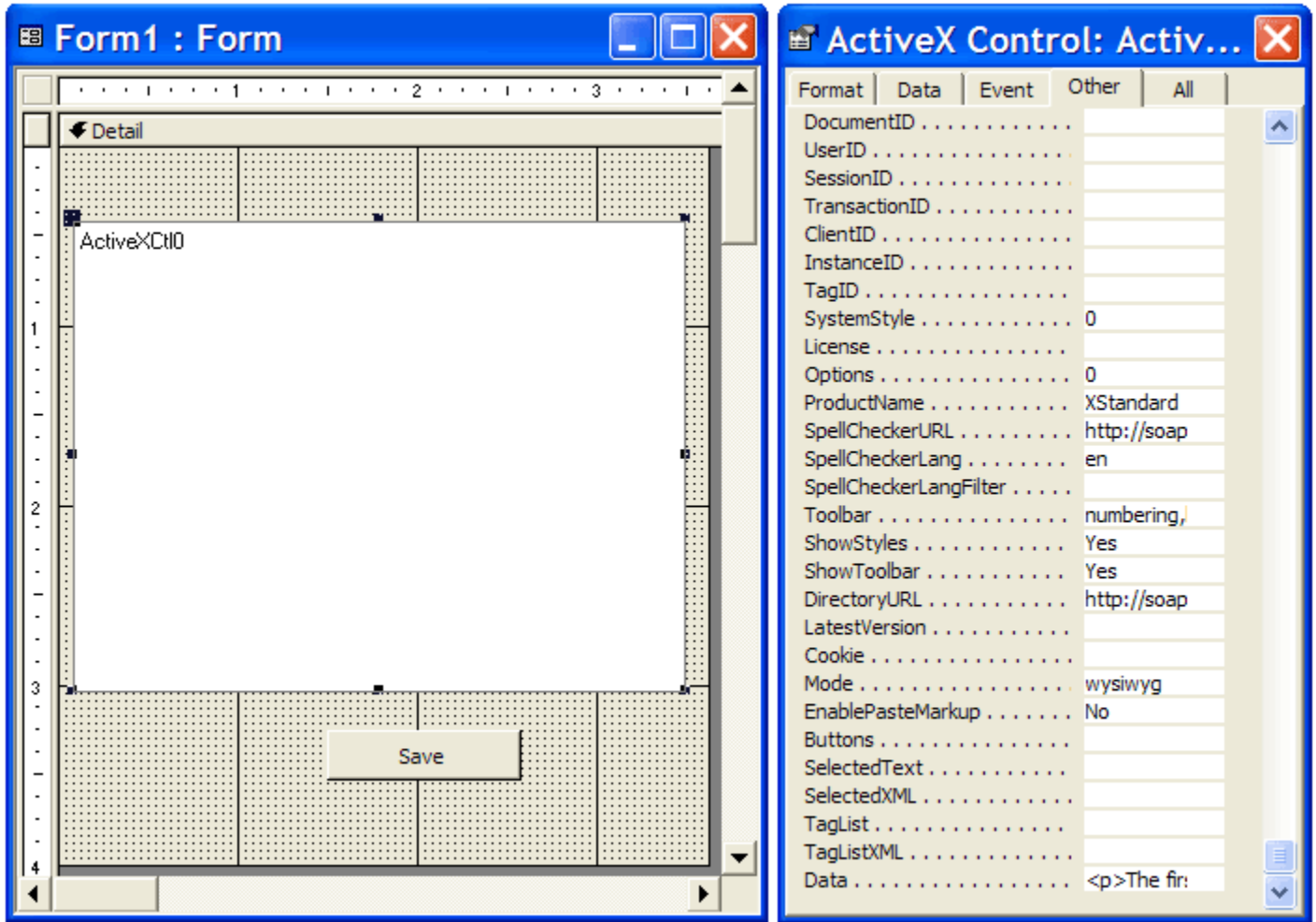


3. From the menu bar, select `Insert > ActiveX Control...` which will bring up a list of ActiveX components registered on your computer as shown in the screenshot below.

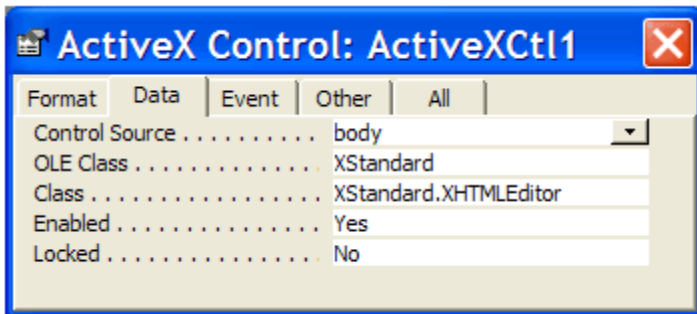


Scroll to the bottom of the list, select "XStandard" and press the `OK` button. The XStandard control will be added to the form.

- Stretch the editor to desired size. To configure the editor, select it by clicking on it and a list of available properties will be displayed in the "Properties" window as shown in the screenshot below.



- To bind a database field to the editor, click on the "Data" tab in the "Properties" window and specify the database field in the "Control Source" property as shown in the screenshot below.



## Tips

When you need to programmatically get or set the value (XHTML) in the editor, use the `Data` property instead of the `Value` property.

Access does not pass the BACKSPACE key to the editor. Use the following workaround:

- Set the form's "Key Preview" property to "Yes".
- Include the following KeyDown event subroutine in the form's code module:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeyBack And Shift = 0 Then
KeyCode = 0
SendKeys "+{BS}"
End If
End Sub
```



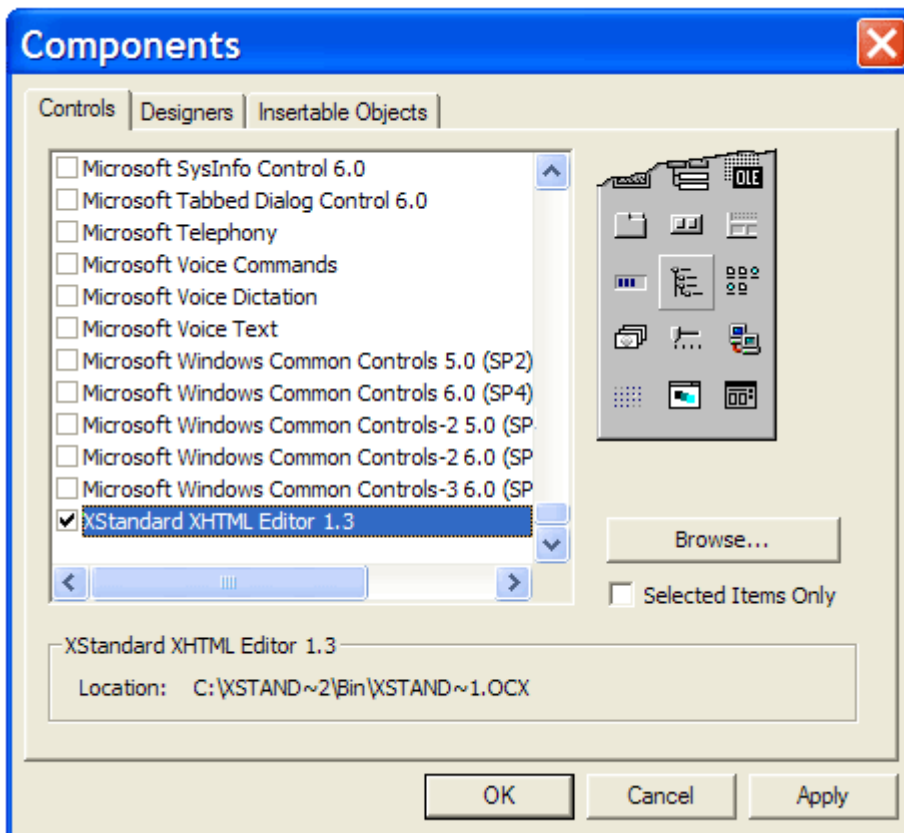
```
End If
End Sub
```

Pressing Undo button in Access may clear the contents of the editor. Use the following code to let users control the undo operation:

```
Private Sub Form_Undo(Cancel As Integer)
If MsgBox("This operation may clear contents in the editor. Do you want to cancel the Undo?",vbYesNo) = vbYes Then
Cancel = True
Else
Cancel = False
End If
End Sub
```

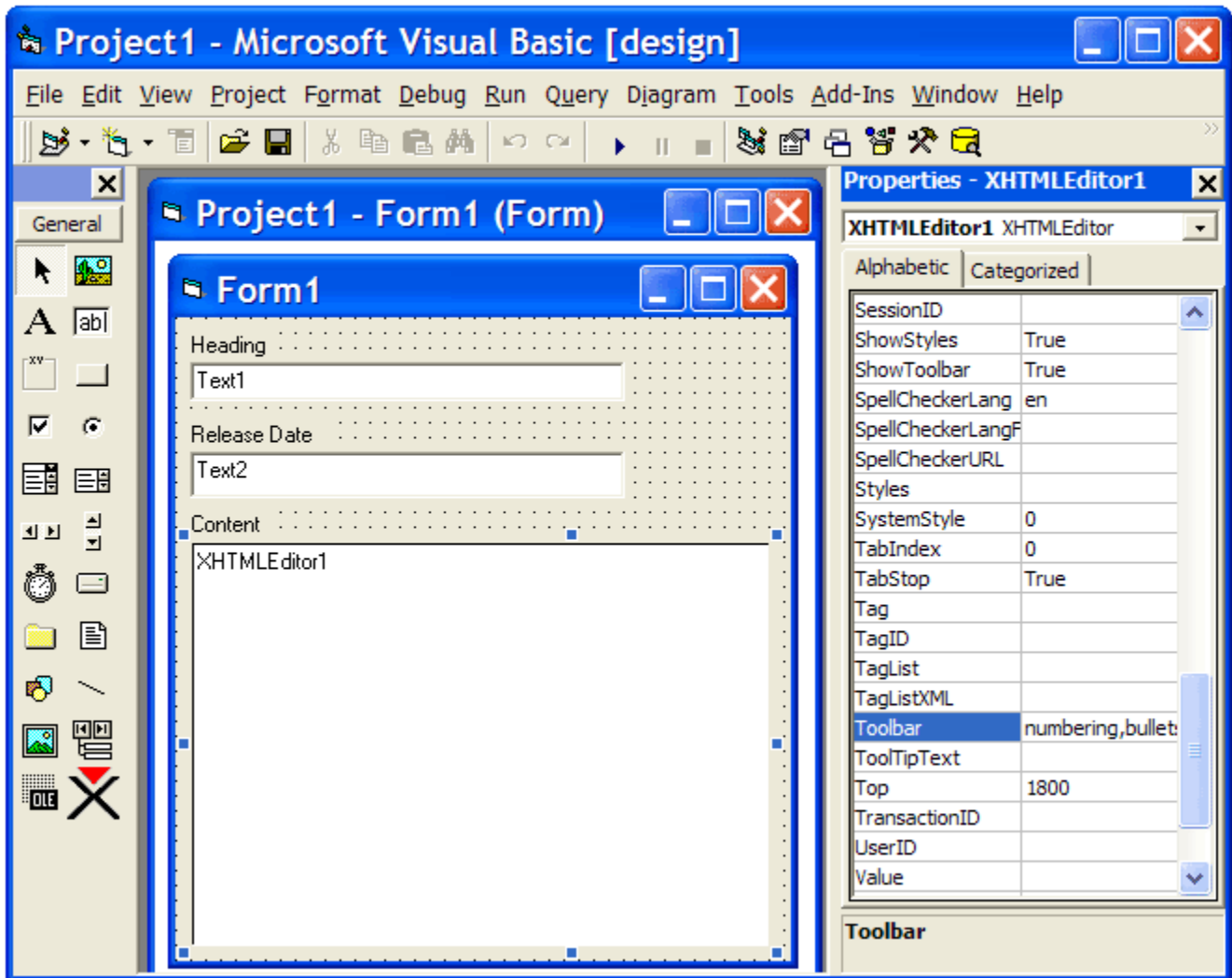
## Visual Basic 6

1. Start Visual Basic and create a new EXE project.
2. From the menu bar, select `Project > Components...` This will bring up a list of components registered on your computer as shown in the screen shot below.



Scroll to the bottom of the list, select "XStandard XHTML Editor" and press the `OK` button. The XStandard control will now be visible on the Toolbox.

3. Select the XStandard control from the Toolbox and place it on a form. Use the "Properties Window" to configure the editor as shown in the screenshot below.



## Tips

Put data into the editor via the `.Value` property. For example:

```
XHTMLEditor1.Value = "<p>Hello World</p>"
```

Use the `.Value` property to get the data from the editor. For example:

```
MsgBox XHTMLEditor1.Value
```

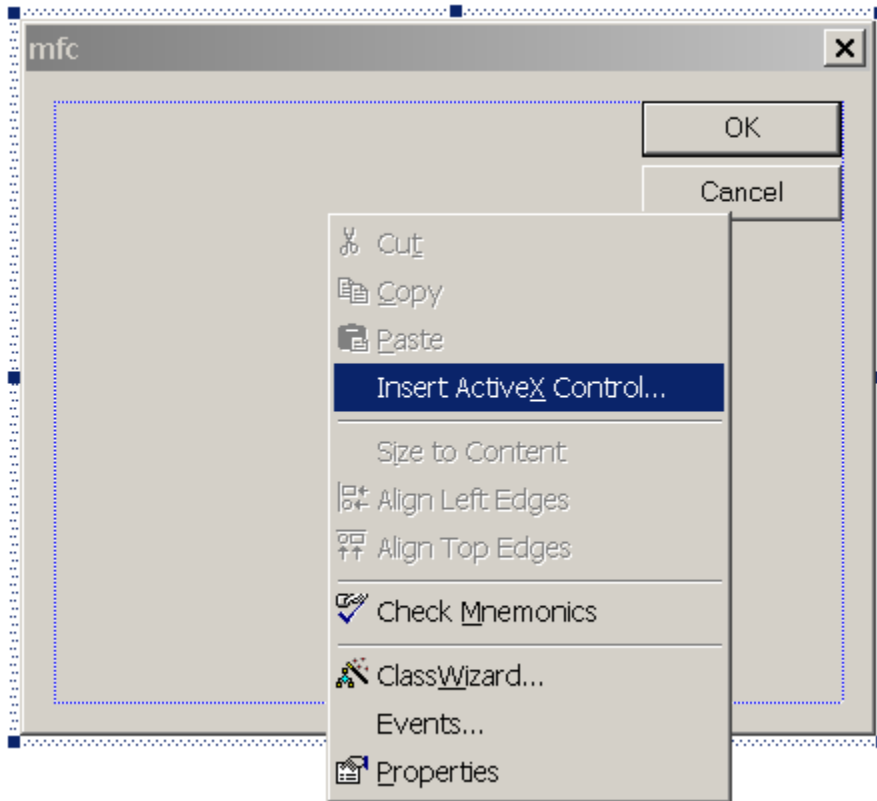
You can store the XML file for the Styles drop-down list, CSS, XML file for the toolbar buttons and the license file in "VB Resources". You can then programmatically reference these files. For example:

```
XHTMLEditor1.Styles = StrConv(LoadResData(101, "CUSTOM"), vbUnicode)
```

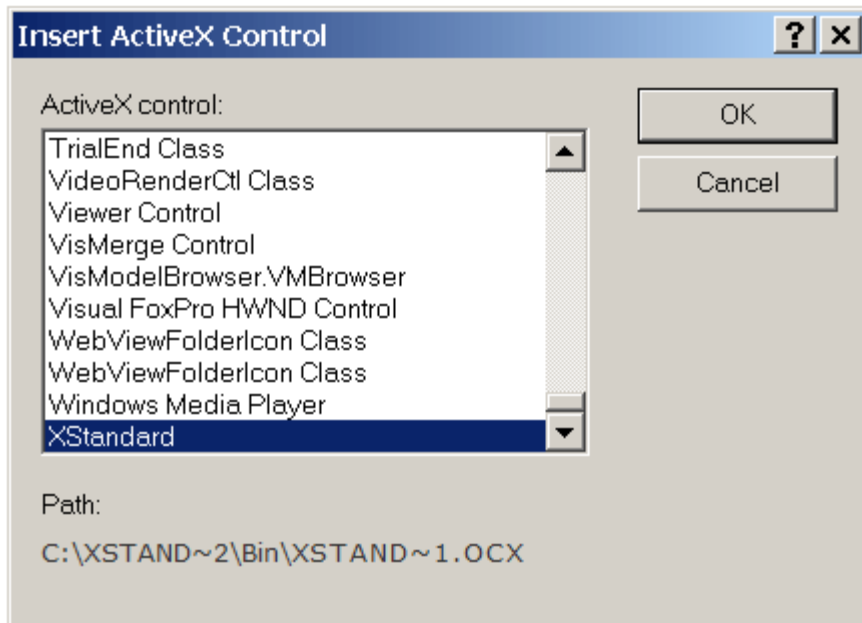
## Visual C++ 6

1. Start Visual C++ and create a new MFC project.

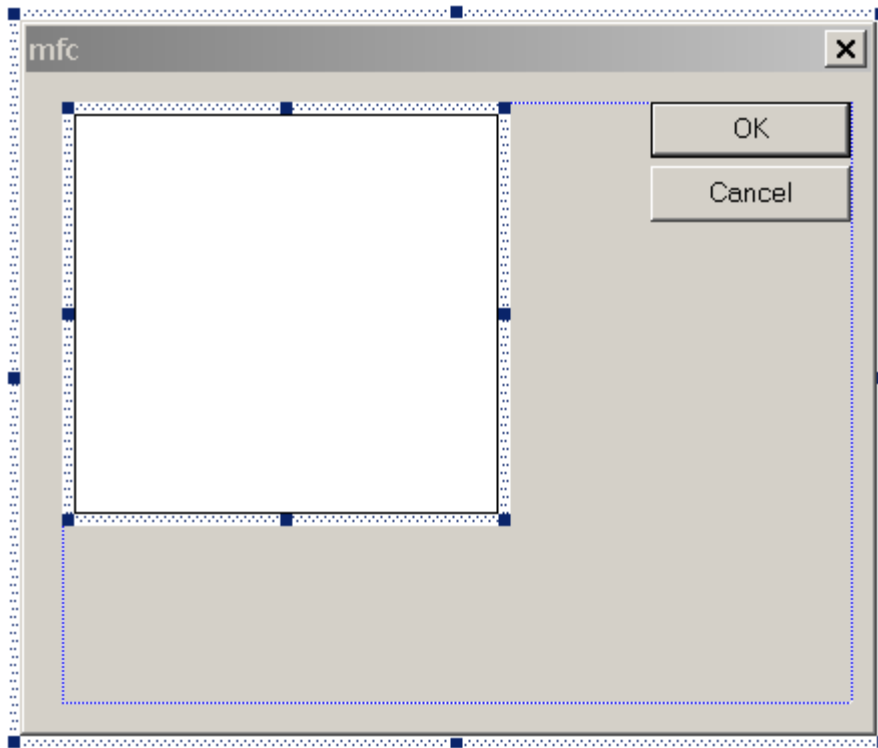
- Right click over a dialog box where you would like to insert the editor and select `Insert ActiveX Control...` as show in the screen shot below.



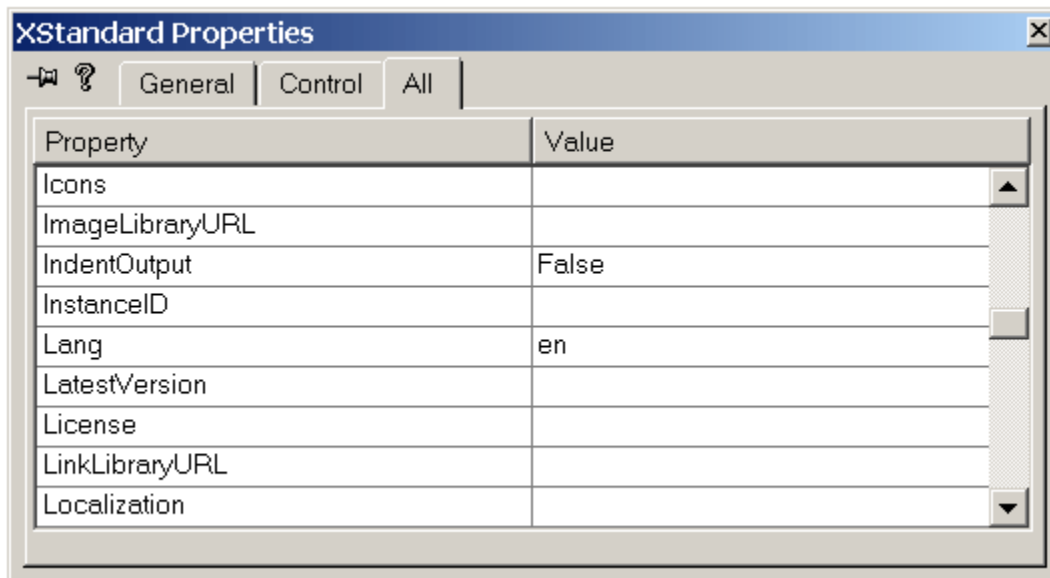
- In the "Insert ActiveX Control" dialog box, select "XStandard" towards the bottom of the list as show in the screen show below.



4. The editor will be inserted into the dialog box as shown in the screen shot below.



5. To edit the editor's properties, right click over the editor and select `Properties`. "XStandard Properties" dialog box will display as shown in the screen shot below.



## Note

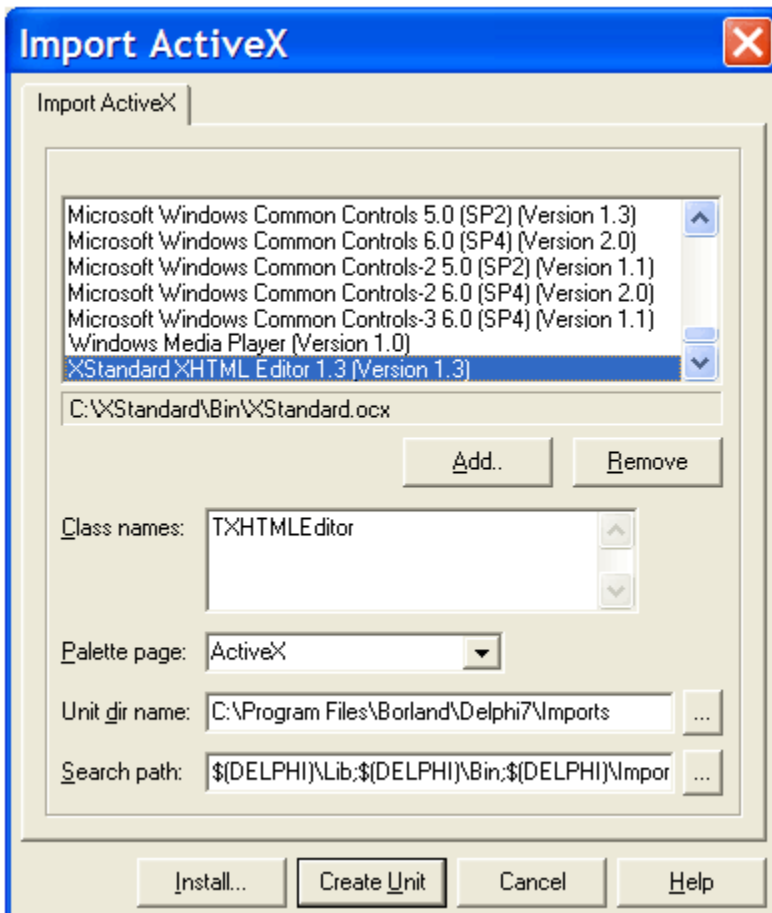
XStandard can also be created dynamically at run-time. See code below as an example:

```
Crect rect;  
rect.left = 20;  
rect.right = 500;  
rect.top = 60;  
rect.bottom = 300;  
static CLSID const clsid = { 0x0EED7206, 0x1661, 0x11d7, { 0x84, 0xa3, 0x0, 0x60, 0x67,  
0x44, 0x83, 0x1d } };  
//this is definition of m_Editor:  
//CWnd *m_Editor;
```

```
m_Editor = new CWnd;  
m_Editor->CreateControl( clsid, "xxx", WS_CHILD | WS_VISIBLE, rect, this, 12346 );  
m_Editor->ModifyStyle(0, WS_CLIPSIBLINGS, 0);  
m_Editor->ShowWindow( SW_SHOW );  
//set property here.  
//set XStandard's "Value" property, its dispid is 1.  
m_Editor->SetProperty( 1,VT_BSTR, "<h1>Hello World!</h1>" );
```

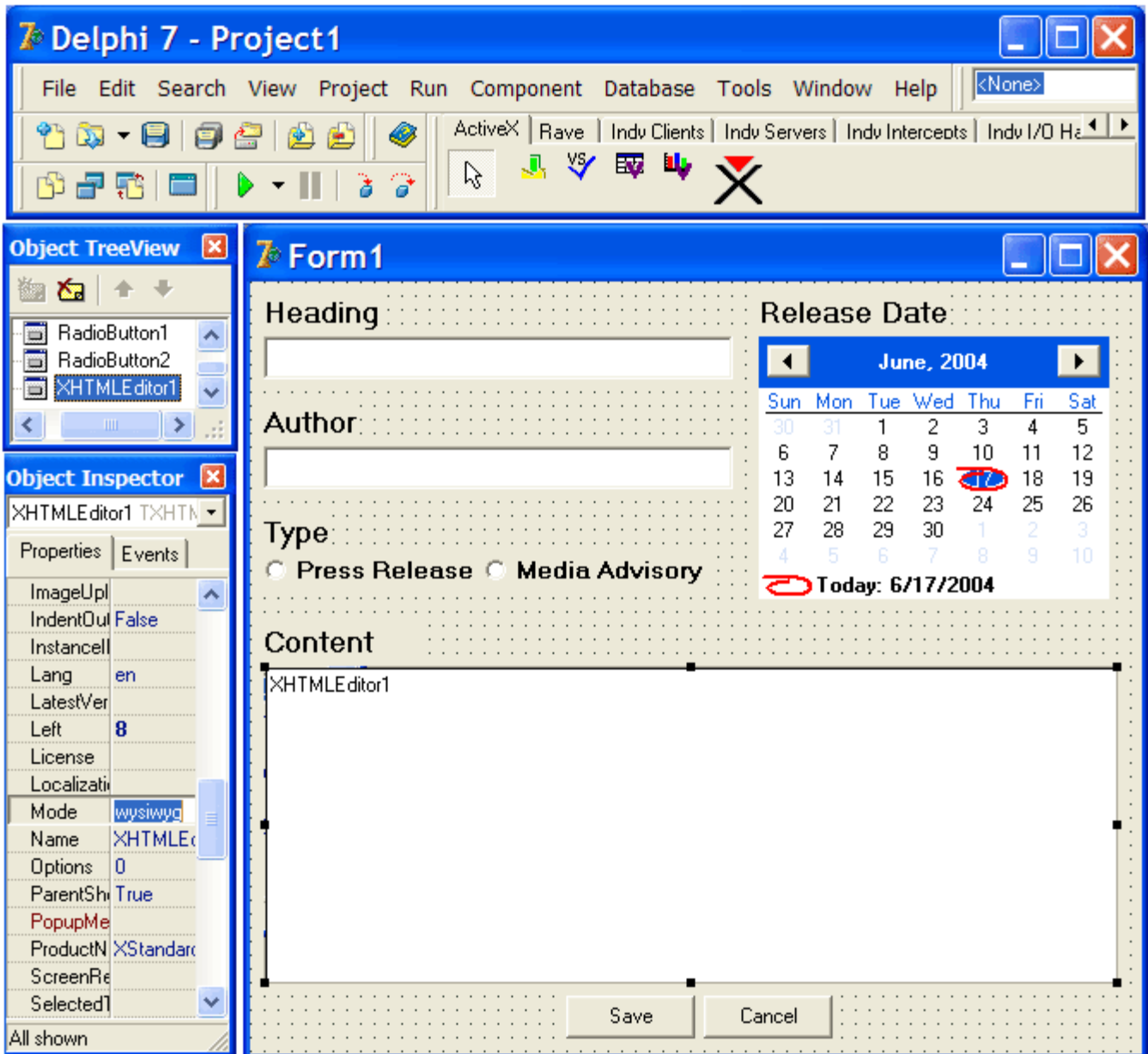
## Delphi 7

1. Start Borland Delphi and create a new Form.
2. This step only needs to be performed the first time using XStandard from Delphi, and may be skipped in the future. From the menu bar, select `Component > Import ActiveX Control...` This will bring up a list of components registered on your computer as shown in the screenshot below.



Scroll to the bottom of the list, select "XStandard XHTML Editor" and press the `Install...` button, then `Create Unit` button.

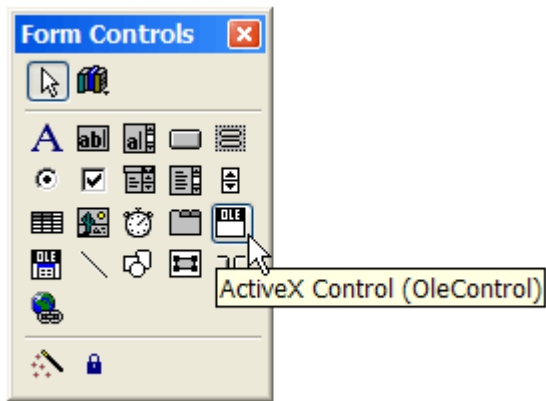
3. Select the ActiveX tab in the toolbar, click on the XStandard icons and then place it on the Form. Use the "Object Inspector" to configure the editor as shown in the screenshot below.



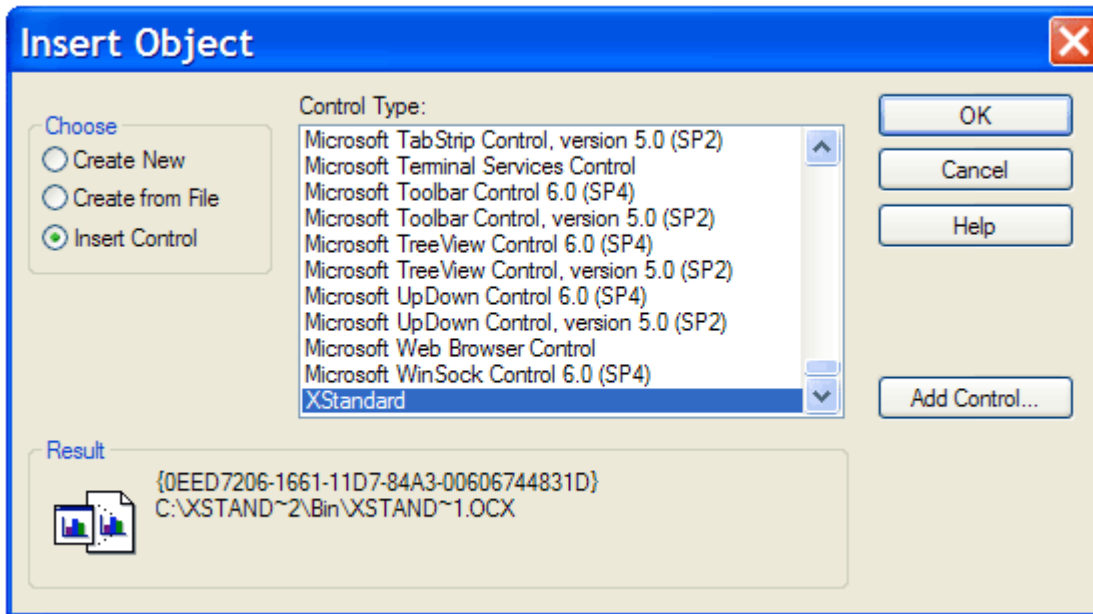
## Visual FoxPro 9

1. Start Visual FoxPro and create a new project.
2. In "Project Manager" window, select that "Documents" tag, select "Forms" and click on the **New...** button in order to create a new form.

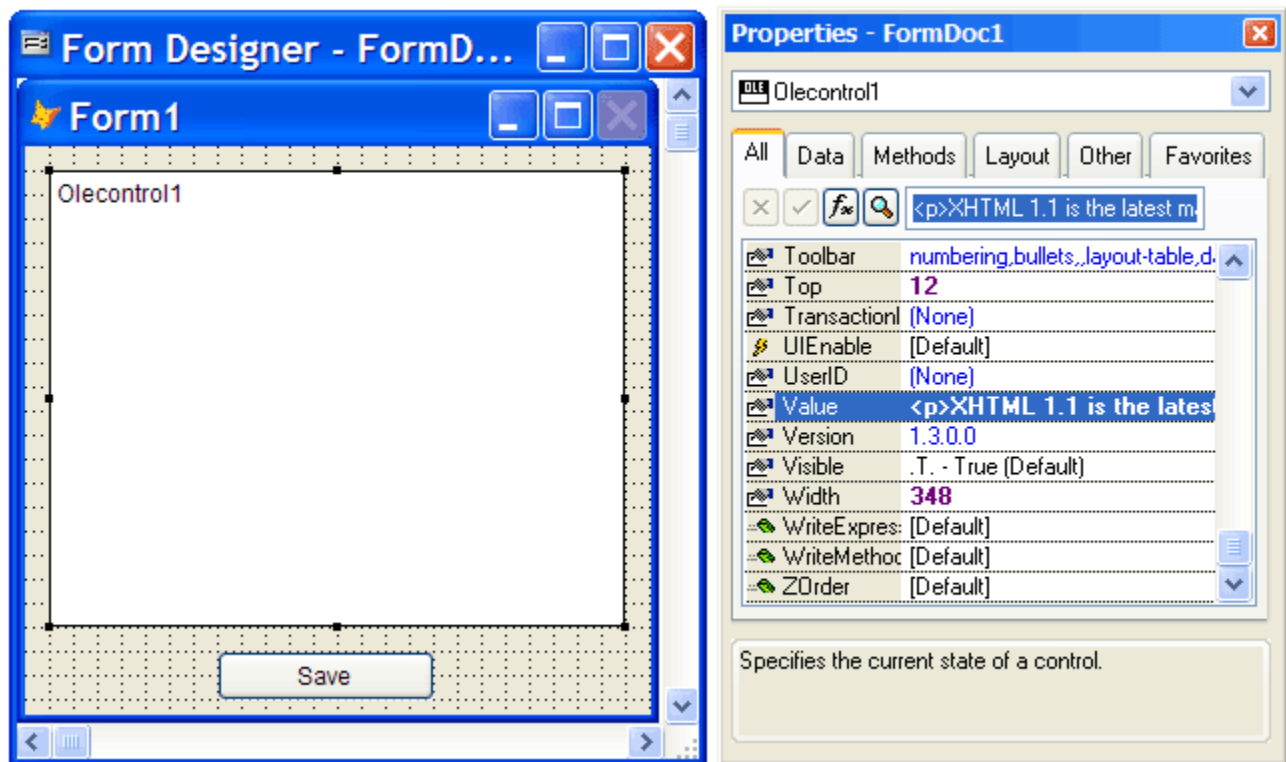
3. In the "Form Controls Toolbar" select the **OLE** button as shown in the screenshot below.



4. With the "OLE" button selected, click on the form to bring up "Insert Object" dialog box show in the screenshot below. Scroll to the bottom of the list, select "XStandard" and press the **OK** button.



5. Stretch the editor to desired size. To configure the editor, select it by clicking on it and a list of available properties will be displayed in the "Properties" window as shown in the screenshot below.



## Accessibility

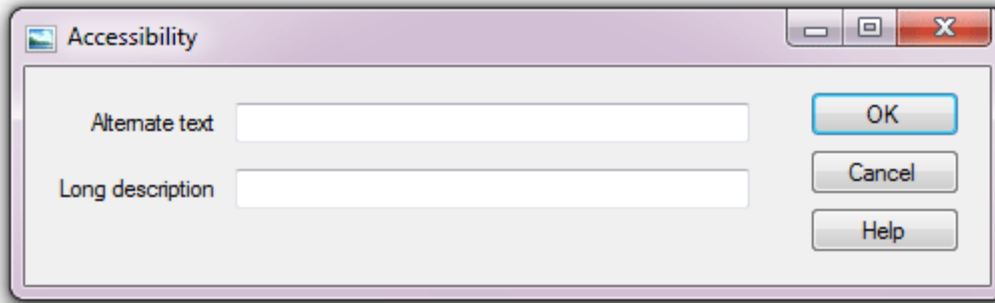
- [Making content containing images accessible](#)
  - [The function of alternate text](#)
  - [Images As Text makes authoring alternate text simple](#)
  - [Images As Text makes editing alternate text equally simple](#)
  - [Images As Text exposes alternate text to find/replace and spell checking](#)
  - [Benefits of Images As Text](#)
  - [Decorative versus non-decorative images](#)
- [Removing the "noise" from markup](#)
- [Encouraging correct use of markup](#)
- [Supporting relative units of length](#)
- [Distinguishing between data and layout tables](#)
- [Supporting abbreviations](#)
- [Exposing content to a Screen Reader Preview](#)
- [Keyboard-accessible interface](#)
- [Keyboard shortcuts](#)
- [Complying with accessibility guidelines](#)

## Making content containing images accessible

### *The function of alternate text*


The function of alternate text is to make content that contains images understandable when images cannot be seen, either by users with limited vision, by search engines, or when users turn off image rendering in their browser. To fulfill its function therefore, alternate text must take into consideration text surrounding images, so that the alternate text works with surrounding content when the images cannot be seen. This is difficult to do in a pop-up dialog box interface such as the following one, which does not display any adjacent content at all:

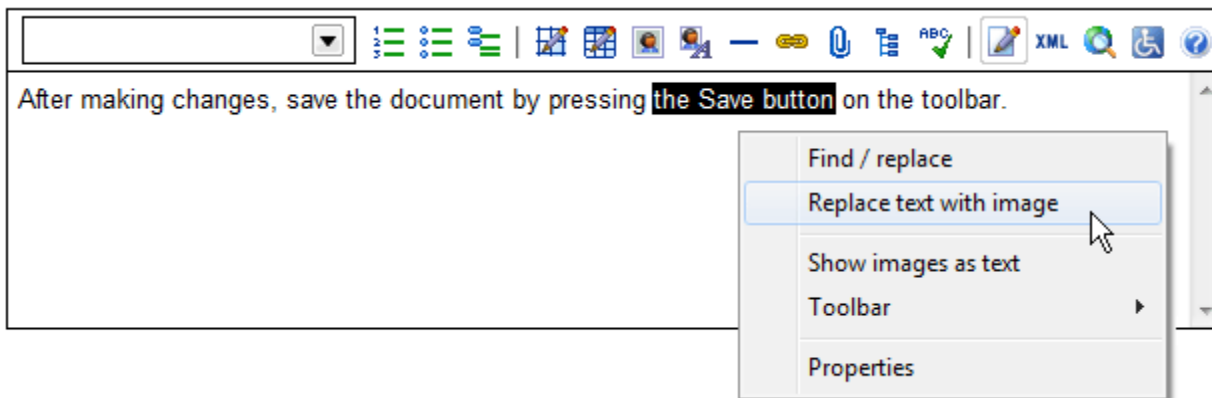




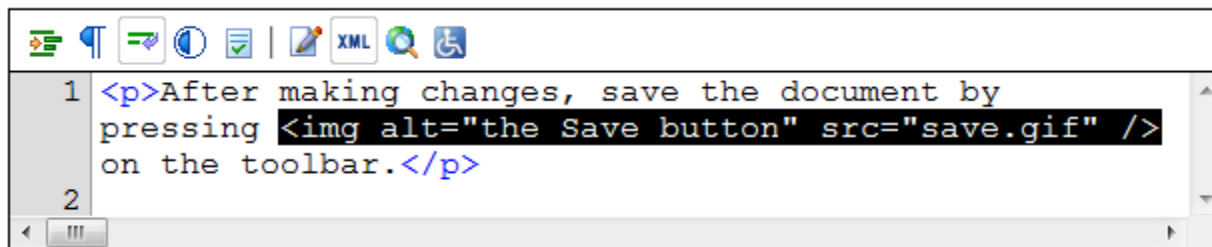
### ***"Images As Text" makes authoring alternate text simple***

XStandard's "Images As Text" feature lets authors create and edit alternate text directly in the document. By displaying alternate text within surrounding content, it becomes clear what the function of alternate text is, and what alternate text will work in a given context.

A foolproof method of authoring alternate text using "Images As Text" is shown in the following screen shot. The author highlights text in the document, then selects "Replace text with image" from the context menu, or by pressing  on the toolbar.




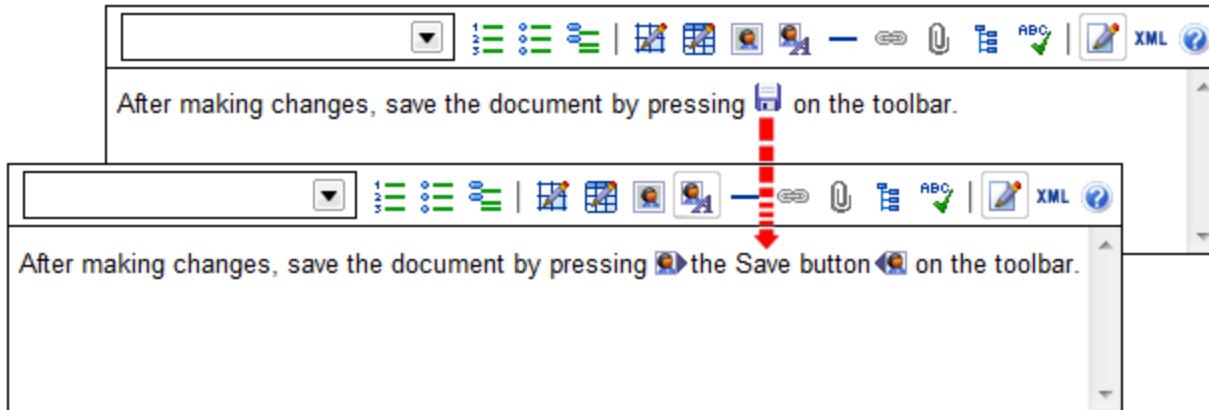
The author then selects an image, which replaces the highlighted text in the document. The highlighted text now becomes the alternate text for the image:



When the document is read as text only, alternate text authored in this way is sure to read perfectly within the surrounding content.

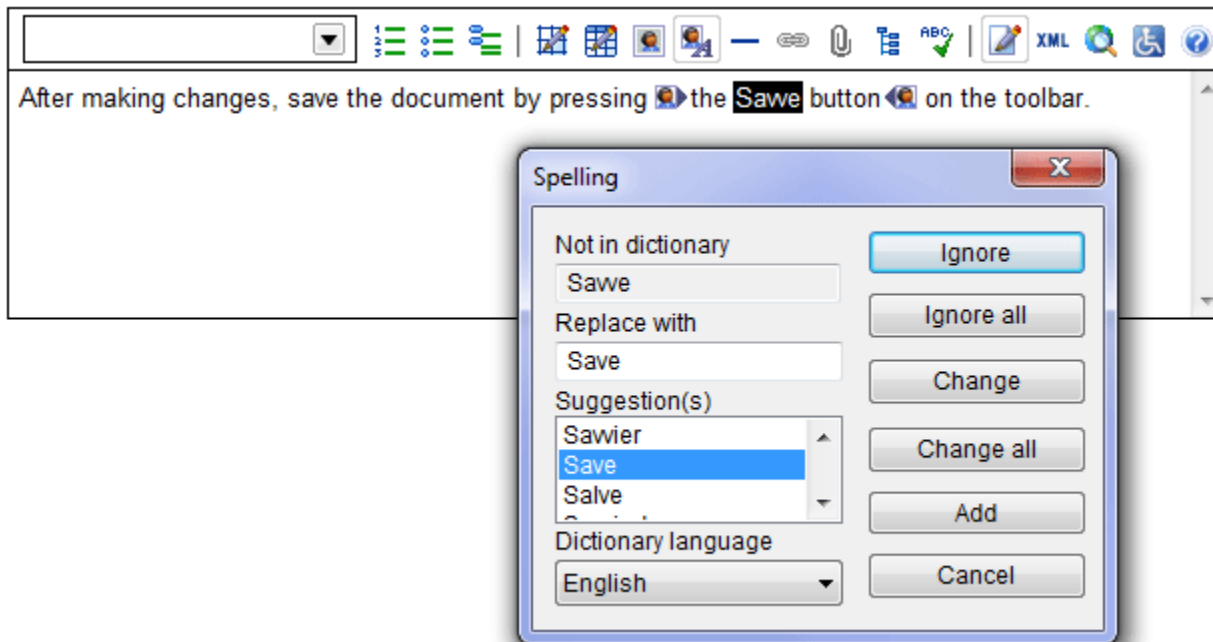
### ***Images As Text makes editing alternate text equally simple***

Images As Text allows alternate text to be edited right in the document. In the following screen shot, the author selects "Show images as text" from the context menu or presses  on the toolbar. This reveals the current alternate text between image markers, where it can be edited directly in the document, just like surrounding content.



## ***Images As Text exposes alternate text to find/replace and spell checking***

When performing editing operations such as find/replace or spell checking, the editor automatically switches to Images As Text mode. This includes alternate text in find/replace and spell checking operations.



## ***Benefits of Images As Text***

- The Images As Text feature significantly reduces the skill required to author appropriate alternate text. Composing and editing alternate text directly in the document, rather than in a pop-up dialog box, is easy and produces better results. It makes it clear what alternate text will work and what won't.
- Some authors struggle to understand what alternate text is. This method of authoring alternate text "in-context" makes it immediately evident to authors what the function of alternate text is: to replace images with text that reads coherently within the content surrounding images.
- Content authors now experience and appreciate alternate text as a living part of the document, not as remote, abstracted background information that is only vaguely related to an image.
- Alternate text now fits perfectly into the flow of the document, ensuring the document makes sense when read as text only, or with images.
- Alternate text is for the first time exposed to processing by popular editing features such as find/replace and spell checking, which improves still further the quality of alternate text.

## ***Decorative versus non-decorative images***

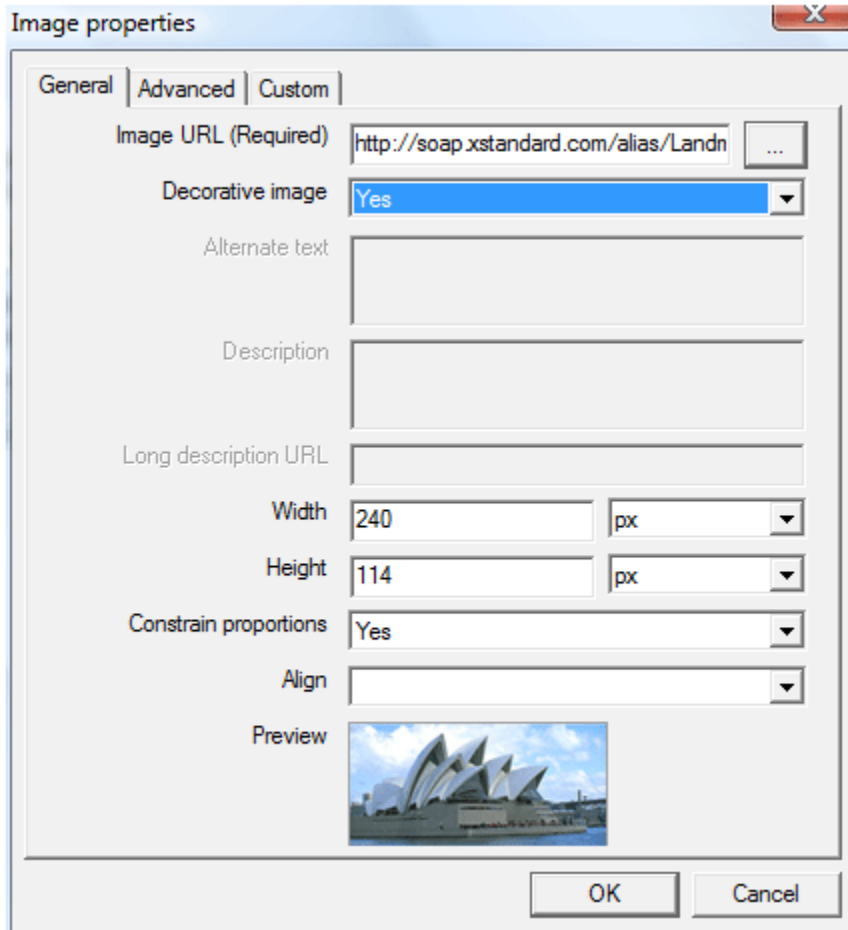
There are two types of images: decorative and non-decorative (also known as informative images). Incorrect use of one or the other can lead to distortions in the meaning of content. Informative images are meaningful to users and so require

alternate text. By contrast, decorative images (such as spacers, bullets, borders, etc.) are merely "eye-candy", convey no semantic meaning at all, and should not therefore use alternate text. To make decorative images invisible to non-visual devices, the setting should be `alt=""`.

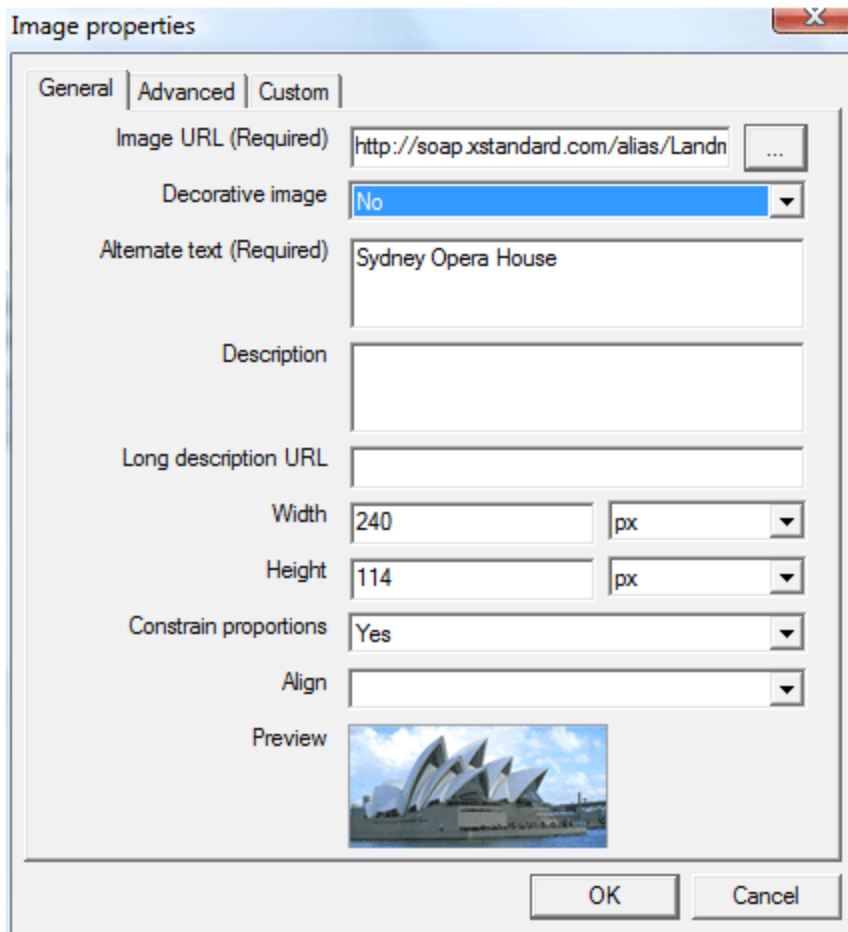
The example below shows alternate text used incorrectly for decorative images. [Listen to the sound file](#) to hear the confusion this creates when the markup is processed by an auditory user-agent such as a screen reader:

```
<p>
Ingredients for black bean soup:<br />
Vegetable broth<br />
Black beans<br />
Crushed tomatoes
</p>
```

Such mistakes become obvious, and are completely avoidable when alternate text is written in the document using the Images As Text feature. However, it is sometimes necessary to edit alternate text in a pop-up dialog box. To avoid the mistake of entering alternate text for decorative images when using pop-up dialog boxes, XStandard prompts the author to identify an image as decorative or non-decorative. If the image is identified as decorative, data is removed from alternate text and the field is disabled, as seen in the following screen shot:



However, if the image is identified as non-decorative, XStandard requires that alternate text be entered for the image before it can be uploaded, as in the following screen shot:



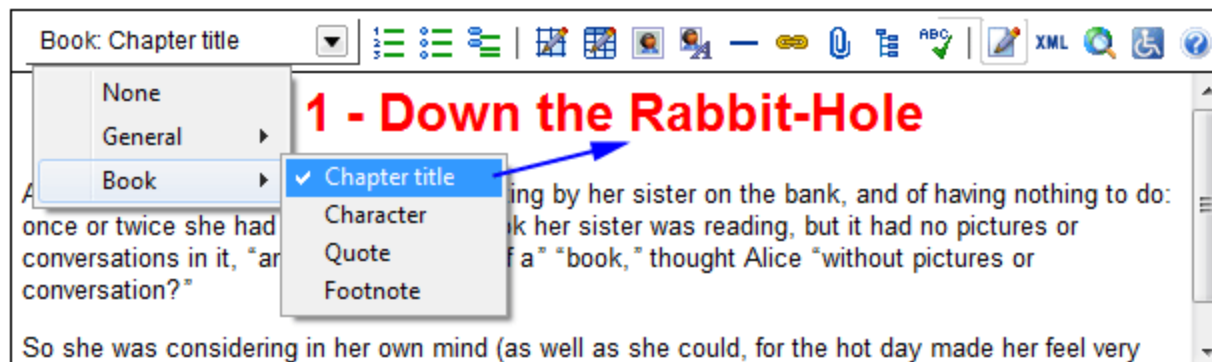
## Removing the "noise" from markup

Deprecated tags, badly formed markup and the fusing together of data and formatting can hinder the processing of content by assistive technologies. XStandard eliminates markup "noise" by generating clean, well-formed XHTML that complies with Web Content Accessibility Guidelines (WCAG) developed by W3C, and completely separates data from formatting by permitting formatting only through external or embedded CSS.

For example, headings (`<h1>` to `<h6>`) play a critical role in making content accessible, because they help users of assistive technologies navigate Web pages. Yet most WYSIWYG editors persist in encouraging authors to construct headings using font selectors and color pickers that generate markup noise like this.

```
<font size="4"><span style="font-family:Times;color:red">Chapter 1 - Down the Rabbit-Hole</span></font>
```

The screen shot below shows how XStandard retains the semantic significance of the same heading, using its Styles menu. The Styles menu contains customizable instructions to generate semantic markup and CSS rules are typically attached to this semantic markup to provide formatting for the content. Styles can be given intuitive labels to ensure that authors apply correct and consistent markup:

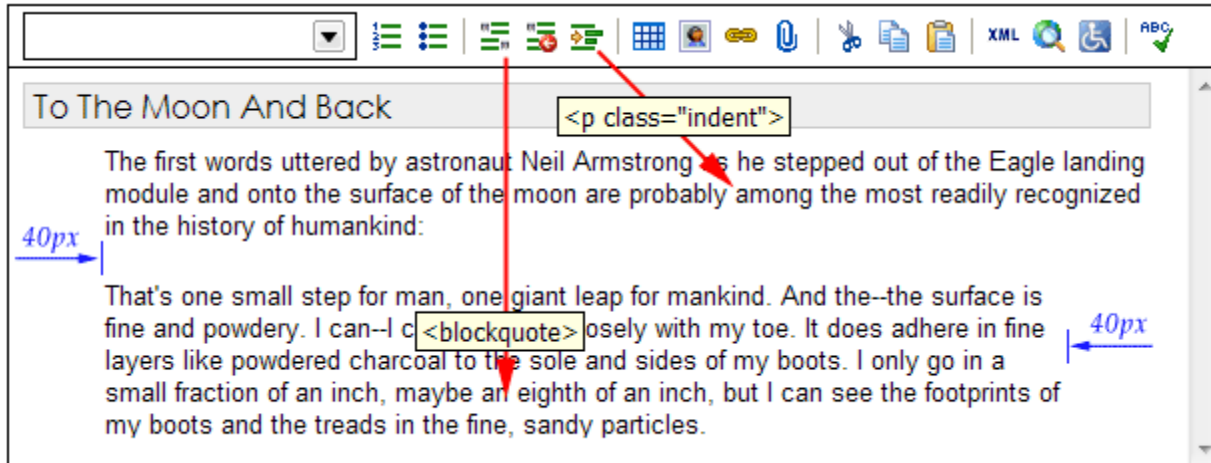


The result this time is semantic markup free of "noise":

<h2>Chapter 1 - Down the Rabbit-Hole</h2>

## Encouraging correct use of markup

Many WYSIWYG editors incorrectly use the `<blockquote>` tag for indenting. This hinders accessibility because it sends the wrong message to assistive technologies. `<blockquote>` should only be used for quotations and indents should be implemented through CSS. The screen shot below shows how XStandard differentiates between block quotes and indents and encourages content creators to use the correct markup for the job.



## Supporting relative units of length

Most browsers allow users to control the size of text displayed on the Web page, but this feature only works if relative units of measure are used in creating Web page content in the first place. XStandard supports both the `em` and `%` units of length.

## Distinguishing between data and layout tables

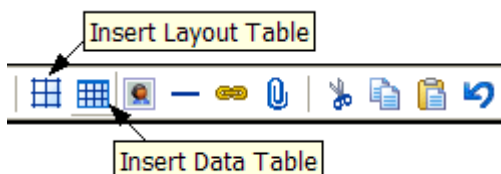
Although tables should be used for displaying data (data tables), until CSS has better support for multi-column organization of content, it is sometimes necessary to use tables for visual layout (layout tables). Data tables and layout tables use different markup. Using the wrong type of table can make content found in tables meaningless to assistive technologies. Below is an example of a data table, where the data in the table is clearly intended to be understood in relation to column headers.

Cups of coffee consumed by each person

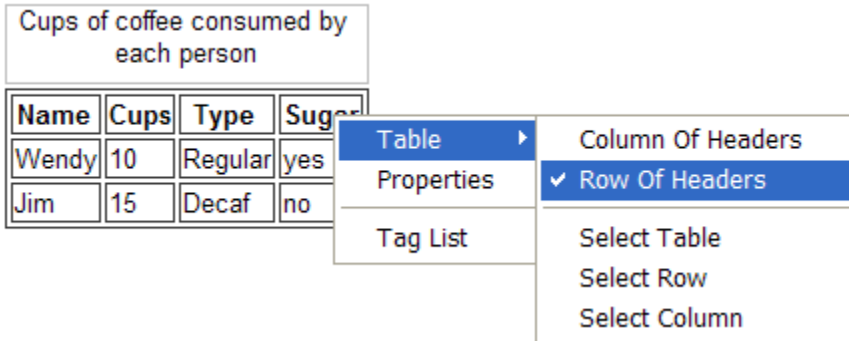
Name	Cups	Type	Sugar
Wendy	10	Regular	yes
Jim	15	Decaf	no

If the markup behind this table does not associate each cell with the appropriate header (as in a data table), the cells will be processed like `<div>` tags by non-visual devices. [Listen](#) to an auditory user-agent "reading" this table as if it is a layout table. Now [listen](#) to the same table as a data table, using correct markup.

To avoid mistaking table types, XStandard offers authors a clear choice between layout tables and data tables, making sure the right table is selected for the job:



XStandard also makes it easy to identify row and column headers, using the Table pop-up menu seen in the screen shot below:

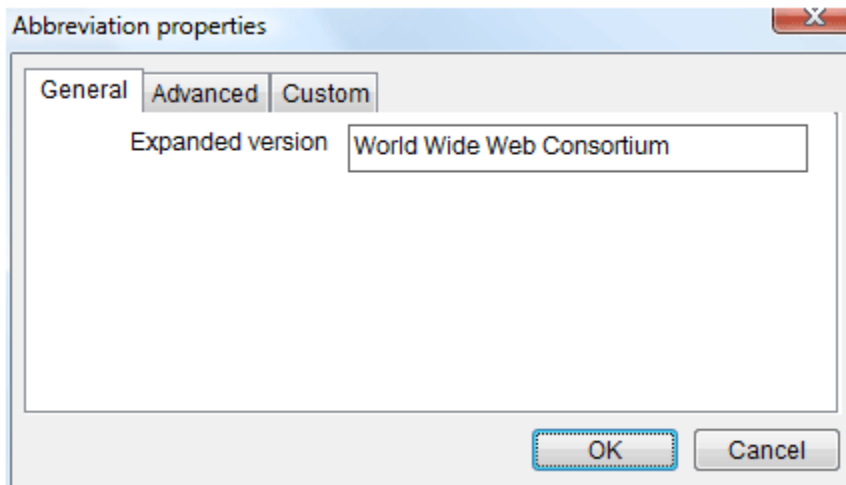


## Supporting abbreviations

Abbreviation tags help auditory devices such as screen readers to correctly pronounce abbreviated words and to render the expanded version of abbreviations. The expanded version of the word is placed in the `title` attribute as shown in these examples.


```
<abbr title="World Wide Web Consortium">W3C</abbr>
<abbr title="Manufacturer">Mfr.</abbr>
<abbr title="5 5 5 - 1 2 3 4">555-1234</abbr>
```

XStandard makes it easy to use abbreviations and acronyms and even prompts authors to enter the "Full Text" for the `title` attribute, as shown in the screen shot below.



The use of "Full Text" for abbreviations and acronyms can better convey the meaning of content to users of assistive technologies, but overuse of "Full Text" can become annoying. As a best practice, XStandard therefore prompts for "Full Text" only if the "Full Text" for an abbreviation has not been entered elsewhere.

## Exposing content to a Screen Reader Preview

The Screen Reader Preview feature, which is activated by pressing  on the toolbar, displays content in the linear manner common to screen readers. This feature helps authors consider accessibility when contributing content and encourages them to correct commonly made mistakes that can hinder accessibility.

As authors manage content through XStandard's WYSIWYG interface, the editor automatically makes it impossible to commit errors in coding that hamper accessibility (such as using deprecated tags), or to make errors of omission (such as forgetting to supply a table Summary). However, if content is entered through "View Source", XStandard catches accessibility errors through the Screen Reader Preview.

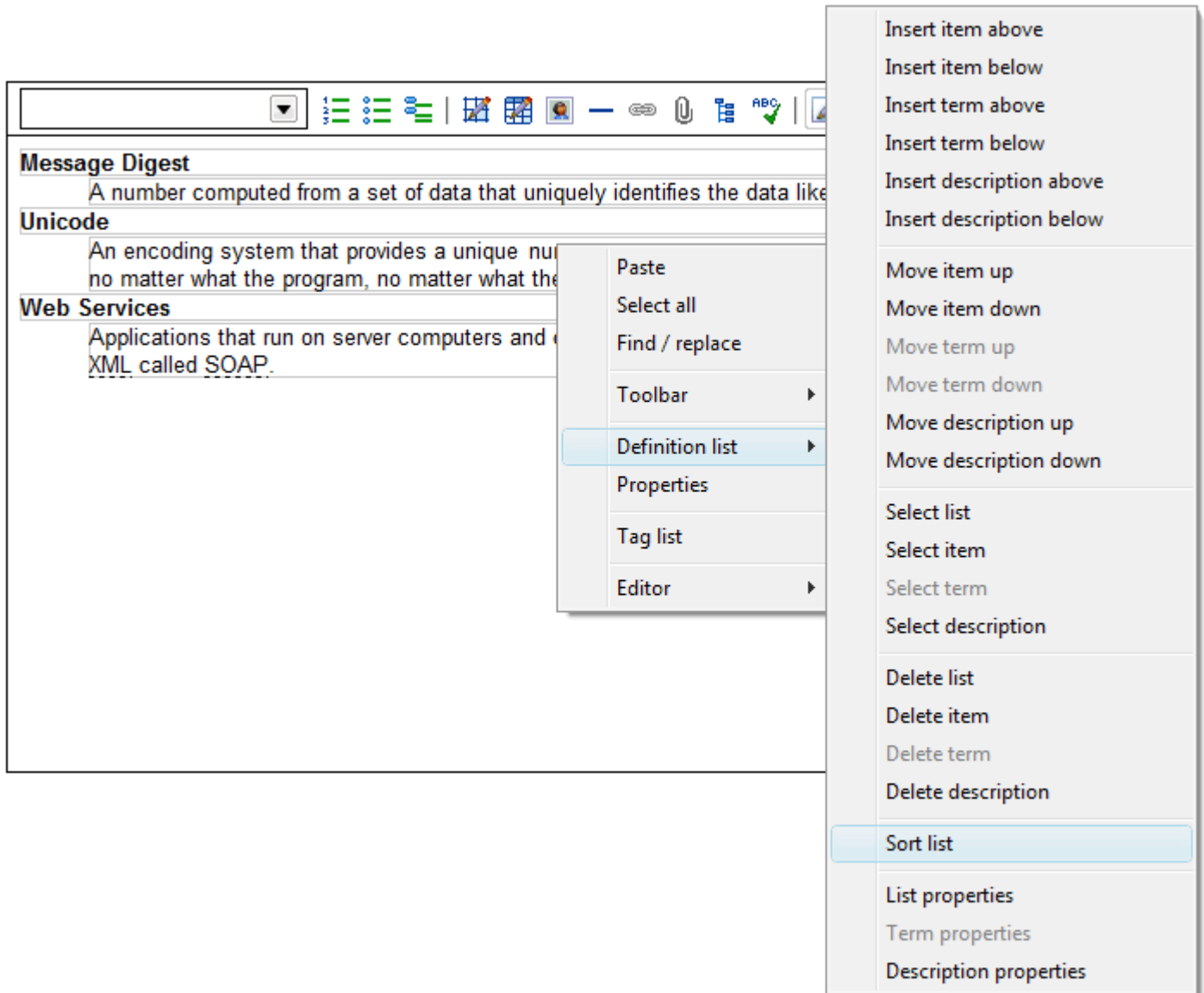
Sample error reports include:

```
This document contains an <b> tag. It is better to use <strong>.
This document contains an <i> tag. It is better to use <em>.
"The alt text for this image is missing."
"The summary text for this table is missing."
```

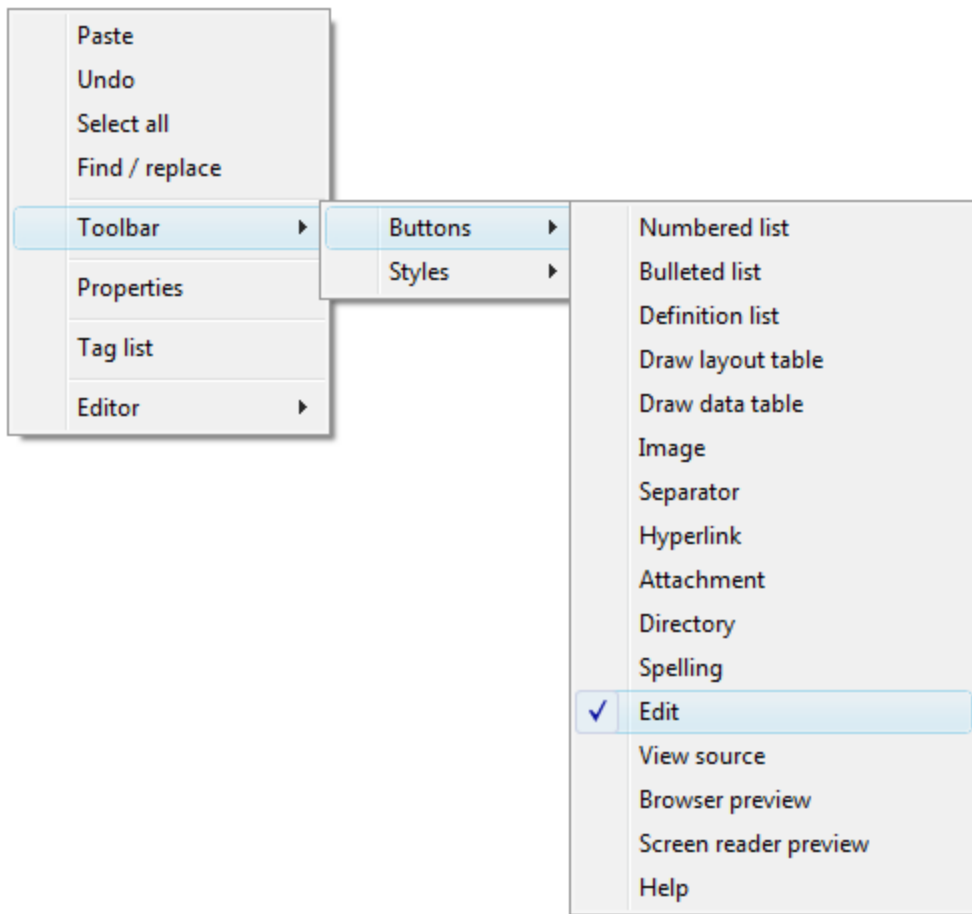
The Screen Reader Preview feature is written in XSLT and can be customized or completely replaced by a specialized version. For instance, XStandard ships with built-in warnings against using ambiguous hyperlink text such as [Click Here](#), but the XSLT can be customized to modify such expressions or to add new ones.

## Keyboard-accessible interface

XStandard is fully keyboard-accessible. This makes it possible to edit even complex constructs such as definition lists. The screen shot below shows editing of definition lists using the keyboard.



The editor's toolbar is also keyboard-accessible via the context menu, as seen in the screen shot below.



## Keyboard shortcuts

Press	To
CTRL + A	Select all.
CTRL + LEFT ARROW	Move cursor to the beginning of the current or previous word.
SHIFT + CTRL + LEFT ARROW	Select text by words.
CTRL + RIGHT ARROW	Move cursor to the beginning of the next word.
SHIFT + CTRL + RIGHT ARROW	Select text by words.
CTRL + UP ARROW	Move cursor to the beginning of the current or previous paragraph or table cell.
CTRL + DOWN ARROW	Move cursor to the beginning of the next paragraph of table cell.
HOME	Move cursor to the beginning of the line.
CTRL + HOME	Move cursor to the top of the document.
END	Move cursor to the end of the line.
CTRL + END	Move cursor to the bottom of the document.
SHIFT + (mouse click or arrow key or HOME or END)	Select or extend the selection.
PAGE UP	Page up.
PAGE DOWN	Page down.
TAB	Move focus to previous control on the form. This feature is supported in most desktop development environments and IE.



Press	To
SHIFT + TAB	Move focus to next control on the form. This feature is supported in most desktop development environments and IE.
CTRL + I	Apply/remove emphasis markup on selected text.
CTRL + B	Apply/remove strong emphasis markup on selected text.
CTRL + K	Bring up the hyperlink properties dialog box when text is selected.
F7	Check spelling.
SHIFT + ENTER	Insert line break.
CTRL + Z	Undo last action.
SHIFT + F10	Display context menu.
ESC	Cancel the current task.
DELETE	Delete selected object.
CTRL + X	Cut
CTRL + C	Copy
CTRL + V	Paste
SHIFT + DELETE	Cut
CTRL + INSERT	Copy
SHIFT + INSERT	Paste

## Complying with accessibility guidelines

XStandard complies with W3C standards, as well as government guidelines and requirements, including the US Section 508 and Canadian Common Look and Feel (CLF) standards.

## Localization

XStandard ships with 22 language interfaces including English, French, German, Spanish, Chinese, Dutch, Italian, Czech, Russian and Swedish. All 22 interface languages are already enabled in the copy of XStandard that you downloaded.

To change the language of the editor's interface, change the localization code in the "Lang" `param` tag. For example, to switch to a German interface use:

```
<param name="Lang" value="de" />
```

## Modifying Localization Files

Making changes to one of the 22 existing interfaces, or creating an entirely new language version is easy.

To modify an existing interface, download and edit the appropriate localization file from the table at the end of this document.

If you are creating an entirely new language version, download the English localization file from the same table. This file has convenient "TODO" place-holders where you will insert the new language version, and "xx" placeholders where you will insert the new language code.

Once you have made changes to an existing language interface, or once you have created a new language version, put the new localization XML file on your Web server, and point to it as follows:

```
<param name="Localization" value="http://myserver/localization.xml" />
```

The localization XML document can store localization for one or several languages at the same time. Multilingual versions such as the one seen in the screenshot below, identify each language by an `xml:lang` attribute, and the `<param>` tag named `Lang` instructs the editor to pick the relevant language version from the localization file.

```

<localization>
  <tooltip>
    <numbering xml:lang="en">Numbering</numbering>
    <numbering xml:lang="fr">Liste numérotée</numbering>
    <numbering xml:lang="cn">编号列表</numbering>

    <bullets xml:lang="en">Bullets</bullets>
    <bullets xml:lang="fr">Liste à puces</bullets>
    <bullets xml:lang="cn">非编号列表</bullets>

    <add-block-quote xml:lang="en">Add Block Quote</add-
    <add-block-quote xml:lang="fr">Ajouter citation mise en
    <add-block-quote xml:lang="cn">跳格</add-block-quote

```

When a single language version of the localization file is used, `xml:lang` attributes are not necessary and the value from the `<param>` tag named `Lang` is ignored.

```

<localization>
  <tooltip>
    <numbering>Numbering</numbering>
    <bullets>Bullets</bullets>
    <add-block-quote>Add Block Quote</add-block-quote>

```

Note: Be sure to save your localization XML file in Unicode (UTF-16 or UTF-8). This will ensure that text in any language will save correctly. If you are editing the localization file in Notepad, when you select "Save As" from the File menu, choose "Unicode" or "UTF-8" as the type of "Encoding".

File name:

Save as type:

Encoding:

Below is a list of available localization files contributed by developers like you. If you would like to contribute a translation, please [contact us](#).

Available Localization Files				
Download	Language Code	XStandard Version	Built-in	Author
<a href="#">English</a>	en	3.0	Yes	Belus Technology
<a href="#">German</a>	de	3.0	Yes	<a href="#">Peter Bässler</a> and <a href="#">Sascha Kleinwächter</a>
<a href="#">Dutch</a>	nl	3.0	Yes	<a href="#">Meint Post</a>
<a href="#">French</a>	fr	3.0	Yes	Monique Glachant
<a href="#">Simplified Chinese</a>	cn	3.0	Yes	You Cheng Pu
<a href="#">Indonesian</a>	id	3.0	Yes	<a href="#">Andronicus Riyono</a>
<a href="#">Javanese</a>	jw	3.0	Yes	Waskito Adi
<a href="#">Russian</a>	ru	3.0	Yes	<a href="#">Oleg Butuzov</a>
<a href="#">Ukrainian</a>	uk	3.0	Yes	<a href="#">Oleg Butuzov</a>
<a href="#">Greek</a>	el	3.0	Yes	<a href="#">Argiris Bendilas</a>

Available Localization Files				
Download	Language Code	XStandard Version	Built-in	Author
<a href="#">Serbian</a>	sr	3.0	Yes	<a href="#">Petar Marić</a>
<a href="#">Serbo-Croatian</a>	sh	3.0	Yes	<a href="#">Petar Marić</a>
<a href="#">Czech</a>	cz	3.0	Yes	<a href="#">Jiří Bureš</a> , <a href="#">Jindřich Hejlík</a> and <a href="#">Radek Hulán</a>
<a href="#">Slovak</a>	sk	3.0	Yes	Lubos Magat
<a href="#">Hungarian</a>	hu	3.0	Yes	Istvan Nagy
<a href="#">Polish</a>	pl	3.0	Yes	Tomek Lisiewicz
<a href="#">Swedish</a>	sv	3.0	Yes	Roger Johansson and <a href="#">Lennart Olsson</a>
<a href="#">Finnish</a>	fi	3.0	Yes	<a href="#">Aki Björklund</a> , <a href="#">Jaana Björklund</a> and <a href="#">Ville Pilvio</a>
<a href="#">Danish</a>	da	3.0	Yes	Jari Berg Jensen and <a href="#">Kenneth Bjørnsholm</a>
<a href="#">Brazilian Portuguese</a>	pt	3.0	Yes	<a href="#">Rodrigo Tisatto</a>
<a href="#">Spanish</a>	es	3.0	Yes	 Camilo Rueda López
<a href="#">Italian</a>	it	3.0	Yes	 <a href="#">Roberto Scano</a>

## Web Services

Web Services are applications that run on the server. They communicate with other computers, using a dialect of XML called SOAP (Simple Object Access Protocol). Typically, business users do not interact directly with Web Services. Instead, they interact with user-friendly programs that themselves communicate with Web Services. XStandard uses Web Services for uploading files from the local computer to the server, for referencing files located on remote servers, for communicating with 3rd-party systems (such as your CMS), and for spell checking.

Web Services functionality is only available in XStandard Pro. The following table describes the type of Web Services that XStandard Pro can communicate with.

Service	Description	Platform
<a href="#">Image Library &amp; Attachment Library</a>	This service is used for building an image and attachment library.	<ul style="list-style-type: none"> <li>ASP/ASP.NET on Windows</li> <li>PHP on Linux / FreeBSD / Windows</li> </ul>
<a href="#">Spell Checker</a>	This service is used for spell checking. Available languages are: English (US, Canadian, British), Danish, German, Spanish, French, Italian, Dutch, Norwegian, Portuguese, Swedish	<ul style="list-style-type: none"> <li>ASP/ASP.NET on Windows</li> <li>PHP on Linux / FreeBSD / Windows</li> </ul>
<a href="#">Directory</a>	This service is used for communicating with 3rd-party systems (such as your CMS) and for inserting code snippets in the editor.	<ul style="list-style-type: none"> <li>ASP/ASP.NET on Windows</li> <li>PHP on Linux / FreeBSD / Windows</li> </ul>
<a href="#">Subdocument</a>	This service is used for rendering subdocuments in the editor instead of custom elements.	<ul style="list-style-type: none"> <li>ASP/ASP.NET on Windows</li> <li>PHP on Linux / FreeBSD / Windows</li> </ul>

To configure XStandard to use Web Services, modify the following params:

```
<param name="ImageLibraryURL" value="http://myserver/imagelibrary.asp" />
<param name="AttachmentLibraryURL" value="http://myserver/attachmentlibrary.asp" />
<param name="SpellCheckerURL" value="http://myserver/spellchecker.asp" />
```

```
<param name="DirectoryURL" value="http://myserver/directory.asp" />
<param name="SubdocumentURL" value="http://myserver/subdocument.asp" />
```

For testing purposes:

- ImageLibrary can be found at: `http://soap.xstandard.com/imagelibrary.aspx`
- AttachmentLibrary can be found at: `http://soap.xstandard.com/attachmentlibrary.aspx`
- SpellChecker can be found at: `http://soap.xstandard.com/spellchecker.aspx`
- Directory can be found at: `http://soap.xstandard.com/directory.aspx`
- Subdocument can be found at: `http://soap.xstandard.com/directory.aspx`

## Spell Checker

- [Overview](#)
- [Requirements](#)
- [ASP Installation](#)
- [ASP.NET Installation](#)
- [PHP Installation](#)
- [Testing](#)
- [Configure XStandard](#)
- [Configure Spell Checker Web Service](#)
- [Notes](#)

### Overview

This service is used for spell checking. Available languages are: English (US, Canadian, British), German, French, Spanish, Italian, Dutch, Danish, Norwegian, Portuguese and Swedish.

### Requirements

- ASP / ASP.NET on Windows
- PHP 4.3.0+ (with pspell module) on Linux / FreeBSD / Windows

### ASP Installation

1. Run the setup program called `x-web-services-asp.exe` Instructions for downloading this program are sent to you when you request download instructions for XStandard Pro.
2. Set "Read & Execute" file permissions on `C:\Program Files\XStandard Web Services` to Everyone.
3. Copy `spellchecker.asp` and `spellchecker.config` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone.

### ASP.NET Installation

- Unzip the file called `x-web-services-asp.zip` Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
- Copy `spellchecker.aspx` and `spellchecker.config` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone. Into a sub-folder, copy the `bin`, `data` and `dict` folders.

### PHP Installation

1. Unzip the file called `x-web-services-php.zip` Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `spellchecker.php` and `spellchecker.config` to a folder on your Web site. Make sure this folder has the broadest possible permission settings. On Unix-based systems, set permissions to `0777`.

### Testing

In a Web browser, navigate to the URL where the SpellChecker service is located. For example: `http://localhost/spellchecker.asp`

If you don't see `Status: Ready` in the browser, the Web Service is not correctly installed. The most common cause of this is incorrectly set file permissions.

### Configure XStandard

To set up XStandard to use the SpellChecker service, modify the following `<param>` tags:

## Property `SpellCheckerURL`

(Available in XStandard Pro)

Absolute URL to a Spell Web Service. For testing purposes, the following URL is

available: `http://soap.xstandard.com/spellchecker.aspx`

For example:

```
<param name="SpellCheckerURL" value="http://soap.xstandard.com/spellchecker.aspx" />
```

## Property `SpellCheckerLangFilter`

(Available in XStandard Pro)

A comma-delimited list of dictionaries that are a sub-set of the dictionaries available for the Spell Web Service. In other words, if the Web Service supports 10 dictionaries but you want XStandard to use only 2, you would specify the two dictionaries in this parameter. For example: `en-ca, fr` The screen shot below shows a selection of dictionaries set using this property.



## Property `SpellCheckerLang`

(Available in XStandard Pro)

A language code to indicate a default dictionary language for spell checking. The default dictionary must be defined in the `spellchecker.config` file for the SpellChecker Web Service. Example: `en-us`

## Configure SpellChecker Web Service

### Spell Checker Config File

The purpose of `spellchecker.config` is to identify which spelling dictionaries are available. If a dictionary is already installed on the server and you wish to make it available, then set `<available>` to `yes` for each `<dictionary>` tag. Below is an example of a `spellchecker.config` file.

```
<spellChecker>
<dictionary>
<name xml:lang="en">English (US)</name>
<name xml:lang="da">Engelsk (US)</name>
<name xml:lang="de">Englisch (US)</name>
<name xml:lang="es">Inglés (US)</name>
<name xml:lang="fr">Anglais (US)</name>
<name xml:lang="it">Inglese (US)</name>
<name xml:lang="nl">Engels (US)</name>
<name xml:lang="pt">Inglês (US)</name>
<name xml:lang="sv">Engelsk (US)</name>
<code>en-us</code>
<jargon>w-accent</jargon>
<size>60</size>
```

```
<stopCheckWordMin>15</stopCheckWordMin>
<stopCheckPercentErrors>50</stopCheckPercentErrors>
<available>yes</available>
</dictionary>
</spellChecker>
```

## Notes

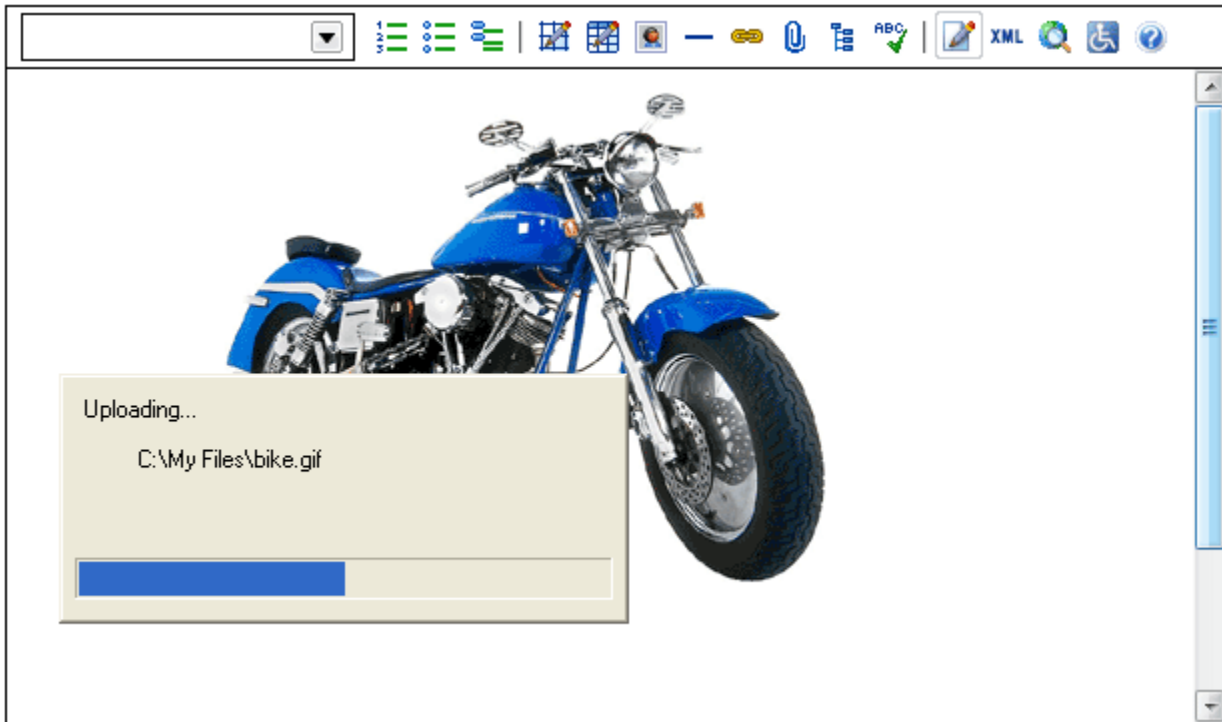
A [Czech version of the PHP SpellChecker Web Service](#) is available. This implementation was kindly contributed by one of our users and is not yet officially supported.

## Image Library & Attachment Library

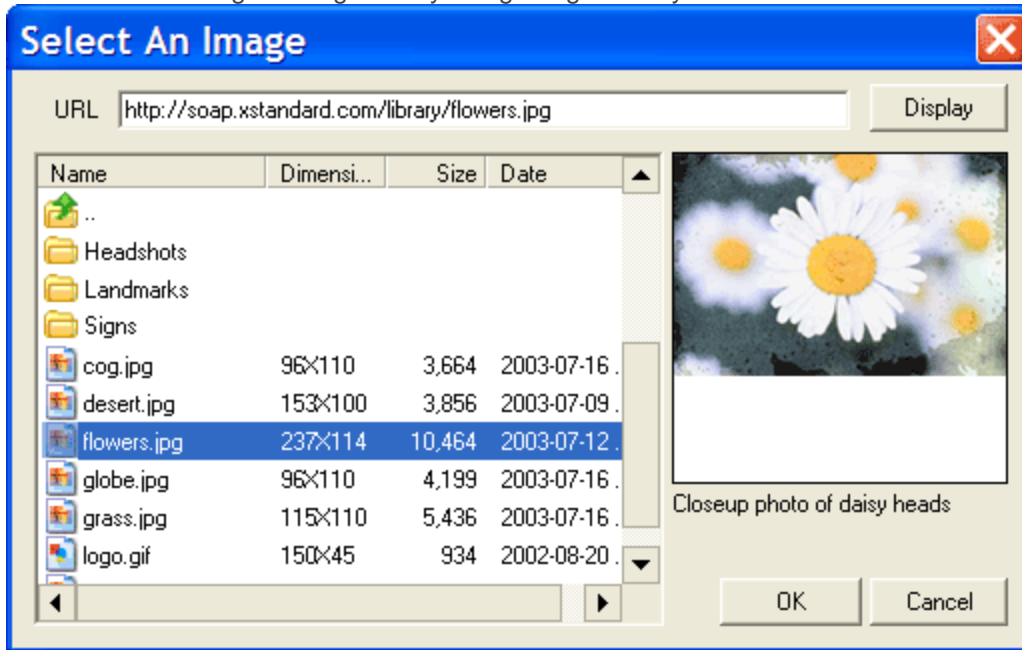
- [Overview](#)
- [Requirements](#)
- [ASP Installation](#)
- [ASP.NET Installation](#)
- [PHP Installation](#)
- [Testing](#)
- [Configure XStandard](#)
- [Configure ImageLibrary And Attachment Library Web Services](#)

## Overview

- These services can be used for uploading files into a library on the server. The screen shot below shows XStandard uploading an image using the Image Library service.



- These services also enable users to select files from a library on the server. The screen shot below shows XStandard browsing an image library using Image Library service.



## Requirements

- ASP / ASP.NET on Windows
- PHP 4.3.0+ (with zlib module) on Linux / FreeBSD / Windows

## ASP Installation

1. Run the setup program called `x-web-services-asp.exe`. Instructions for downloading this program are sent to you when you request download instructions for XStandard Pro.
2. Set "Read & Execute" file permissions on `C:\Program Files\XStandard Web Services` to Everyone.
3. Copy `imagelibrary.asp`, `imagelibrary.config`, `attachmentlibrary.asp` and `attachmentlibrary.config` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone.
4. Set "Read & Write" file permissions on `C:\Temp` to Everyone. This path can be changed in `imagelibrary.asp` and `attachmentlibrary.asp` files.

## ASP.NET Installation

1. Unzip the file called `x-web-services-asp.net.zip`. Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `imagelibrary.asp`, `imagelibrary.config`, `attachmentlibrary.asp` and `attachmentlibrary.config` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone. Into a sub-folder, copy the `bin` folder.
3. Set "Read & Write" file permissions on `C:\Temp` to Everyone. This path can be changed in `imagelibrary.aspx` and `attachmentlibrary.aspx` files.

## PHP Installation

1. Unzip the file called `x-web-services-php.zip`. Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `imagelibrary.php`, `imagelibrary.config`, `attachmentlibrary.php` and `attachmentlibrary.config` to a folder on your Web site. Create a sub-folder called `temp` or modify the PHP script to point to an alternate folder for storing temporary work files. Make sure these folders have the broadest possible permission settings. On Unix-based systems, set permissions to `0777`.

## Testing

In a Web browser, navigate to the URL where the ImageLibrary or AttachmentLibrary service is located. For example: `http://localhost/imagelibrary.asp`

If you don't see `Status: Ready` in the browser, the Web Service is not correctly installed. The most common cause of this is incorrectly set file permissions.

## Configure XStandard

To set up XStandard to use the ImageLibrary and AttachmentLibrary services, modify the following `<param>` tags:

## Property *ImageLibraryURL*

(Available in XStandard Pro)

Absolute URL to an Image Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available: `http://soap.xstandard.com/imagelibrary.aspx`

## Property *AttachmentLibraryURL*

(Available in XStandard Pro)

Absolute URL to an Attachment Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available:

`http://soap.xstandard.com/attachmentlibrary.aspx`

## Property *LinkLibraryURL*

(Available in XStandard Pro)

Absolute URL to a Link Library Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available:

`http://soap.xstandard.com/linklibrary.aspx`

# Configure Image Library And Attachment Library Web Services

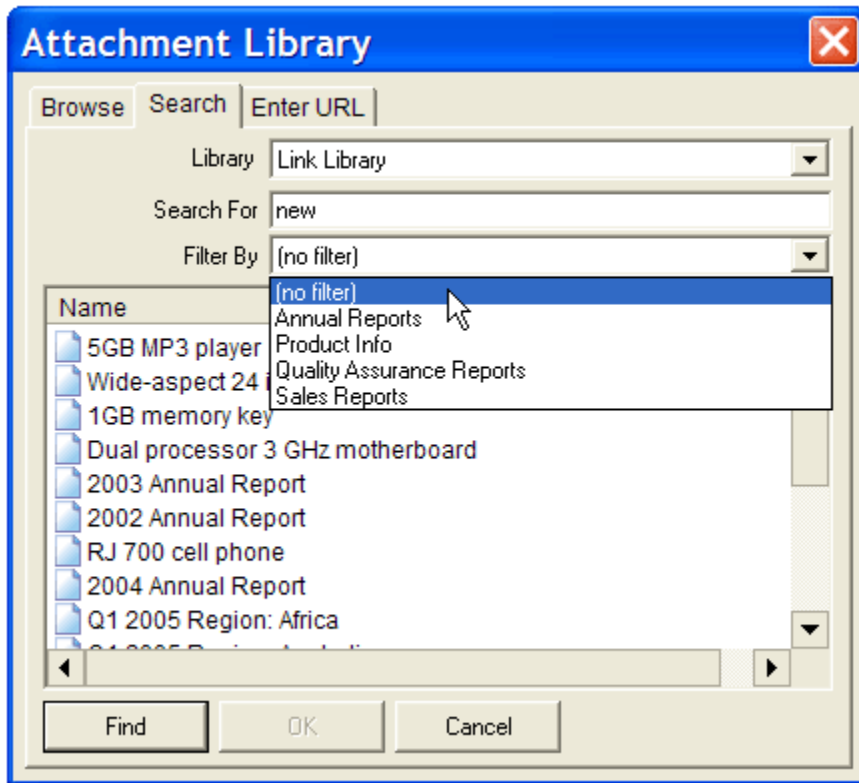
## *Customization*

The following are settings in `imagelibrary.aspx` file. The PHP and ASP versions of this and other services have similar settings.

- The root folder where uploaded images should be saved.  
`string libraryFolder = Server.MapPath(".");`
- The Base URL created for files. This can be a relative or an absolute URL.  
`string baseURL = "images/";`  
Thus, if you upload a file called "file.png" with the above setting, the `src` for the file will be:  
``
- Path to config file.  
`string configFile = Server.MapPath("imagelibrary.config");`
- Temp folder for storing received packets. Make sure file permissions are set to Read/Write for Everyone on this folder.  
`string tempFolder = @"C:\Temp\";`
- A list of accepted file extensions.  
`string acceptedFileTypes = "gif jpeg jpg png bmp";`
- Maximum file upload size in bytes.  
`long maxUploadSize = 512000;`
- Provide date the file was last modified. For large libraries, turning this off can improve performance.  
`bool getDateLastModified = true;`
- Provide file size. For large libraries, turning this off can improve performance.  
`bool getFileSize = true;`
- Provide image dimensions. For large libraries, turning this off can improve performance.  
`bool getImageDimensions = true;`
- Enable browsing of the library.  
`bool libraryBrowseEnabled = true;`
- Enable searching of the library. You will need to customize the Search feature to meet your CMS needs.  
`bool librarySearchEnabled = false;`  
When the Search feature setting is `true`, the editor displays a Search tab similar to that seen in the screenshot

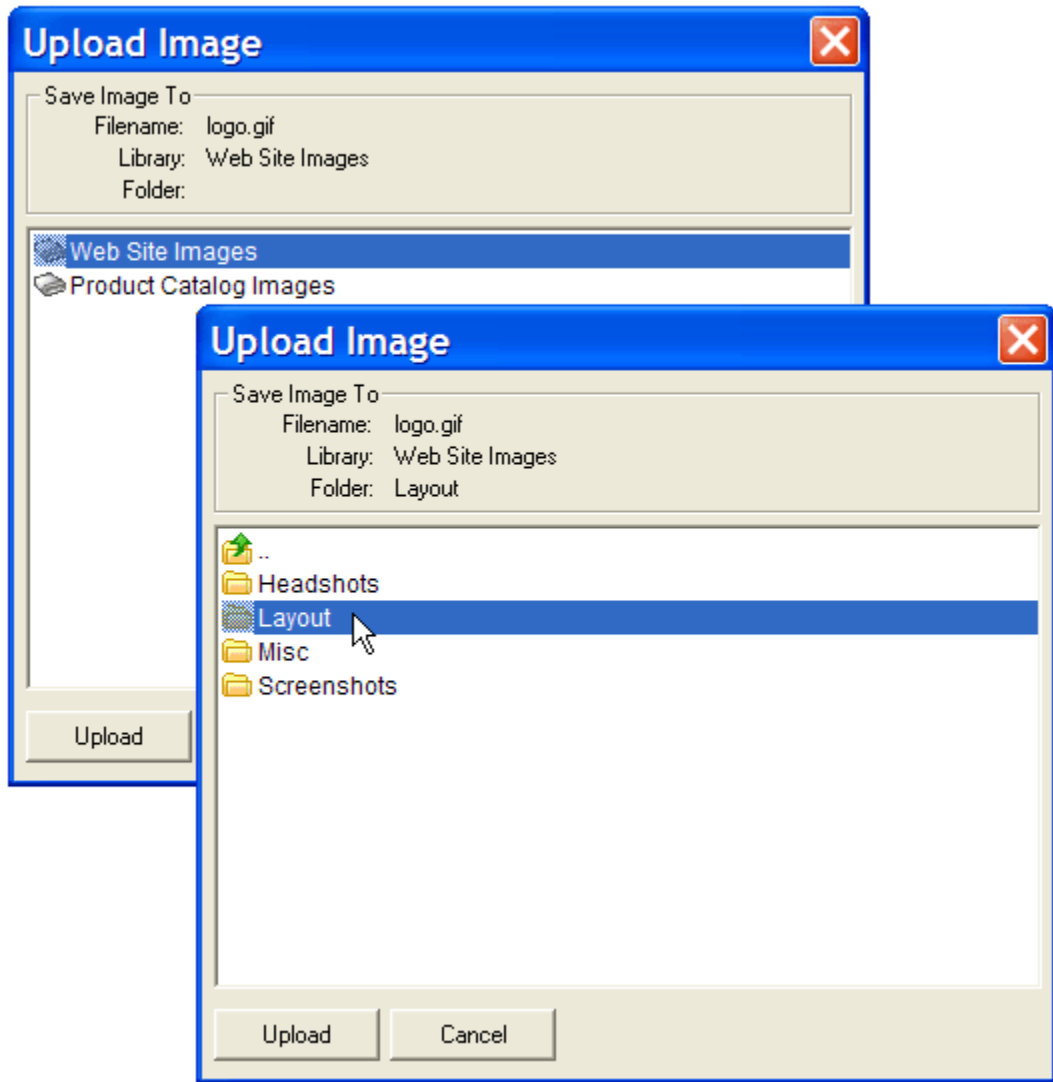


below.



- Enable uploading of files to the root folder.  
`bool libraryUploadToRootContainerEnabled = true;`
- Enable uploading of files to sub-folders.  
`bool libraryUploadToSubContainerEnabled = false;`  
When the sub-folder setting is `true`, uploaded files can be saved to sub-folders, as shown in the screenshot

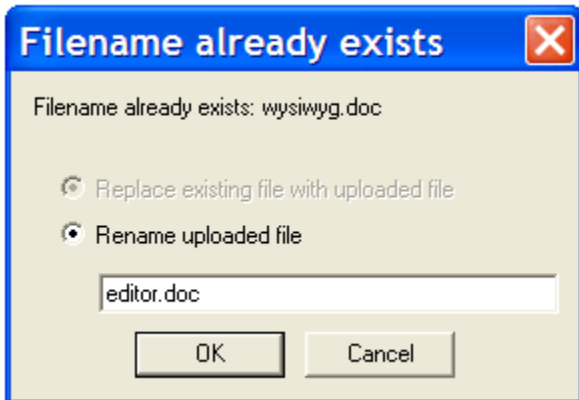
below.



- Enable users to replace existing files with uploaded files.

```
bool libraryUploadReplaceEnabled = true;
```

When set to `false`, the ability to replace existing files with uploaded files is disabled:



### Image Library Config File

The purpose of `imagelibrary.config` is to provide metadata for the image library. Below is an example of an `imagelibrary.config` file. The name of the library is specified in the `<name>` tag. The `<library>` tag can contain multiple `<folder>` and `<image>` definitions. Each folder and image is identified by the `<path>` tag. This is a relative, physical path to the folder or image from the root of the library. For example: `images/headshots/smith.jpg` Note: use `/` as a path separator and ensure that the path does not start or end with `/`. The `<label>` tag can contain a friendly

label for the file or image. Custom icons can be assigned to each folder or image via the `<icon>` tag. This can be the ID of an icon defined in the icons.xml file. A CSS class can be assigned to an image via the `<class>` tag. An image can also be identified as decorative or informative via the `<decorative>` tag. If the value is `no`, values from `<alt>`, `<title>` and `<longdesc>` are used in generating the image markup.

```
<library>
<name>Image Library</name>
<folder>
<path>productphotos</path>
<label>Product Photos</label>
<icon></icon>
<hidden>no</hidden>
</folder>
<image>
<path>file.png</path>
<label></label>
<decorative>no</decorative>
<alt>Alt goes here</alt>
<title>Title text goes here</title>
<longdesc>http://myserver/longdesc.htm</longdesc>
<class></class>
<icon></icon>
<hidden>no</hidden>
</image>
</library>
```

### **Attachment Library Config File**

The purpose of attachmentlibrary.config is to provide metadata for the attachment library. Below is an example of an attachmentlibrary.config file. The name of the library is specified in the `<name>` tag. The `<library>` tag can contain multiple `<folder>` and `<attachment>` definitions. Each folder and attachment is identified by the `<path>` tag. This is a relative, physical path to the folder or file from the root of the library. For example: `docs/reports/Q1sales.pdf` Note: use `/` as a path separator and ensure that the path does not start or end with `/`. The `<label>` tag can contain a friendly label for the folder or file. Custom icons can be assigned to each folder or file via the `<icon>` tag. This can be the ID of an icon defined in the icons.xml file. A CSS class can be assigned to an attachment via the `<class>` tag. The `<newWindow>` tag can be used to generate markup that will open the hyperlinked attachment in a new window.

```
<library>
<name>Attachment Library</name>
<folder>
<path>salesreports</path>
<label>Sales Reports</label>
<icon></icon>
<hidden>no</hidden>
</folder>
<attachment>
<path>file.doc</path>
<label></label>
<newWindow>no</newWindow>
<class></class>
<icon></icon>
<hidden>no</hidden>
</attachment>
</library>
```

## **Directory**

- [Requirements](#)
- [ASP Installation](#)
- [ASP.NET Installation](#)
- [PHP Installation](#)
- [Testing](#)
- [Configure XStandard](#)
- [Configure Directory Web Service](#)
- [Customization For ASP Version](#)
- [Customization For ASP.NET Version](#)
- [Customization For PHP Version](#)

The Directory Web Service creates a gateway through which XStandard is able to browse third-party data stores, then insert the data into the editor at the current cursor position.

## Requirements

- ASP / ASP.NET on Windows
- PHP 4.3.0+ on Linux / FreeBSD / Windows

## ASP Installation

1. Run the setup program called `x-web-services-asp.exe`. Instructions for downloading this program are sent to you when you request download instructions for XStandard Pro.
2. Set "Read & Execute" file permissions on `C:\Program Files\XStandard Web Services` to Everyone.
3. Copy `directory.asp`, `directory.config`, `directory example staff.csv`, `directory-example-product-1.txt`, `directory-example-product-2.txt`, `directory-example-product-3.txt`, `directory-example-product-4.txt`, `directory-xhtml-entities.xml`, `directory-xhtml-latin1.xml`, `directory-xhtml-special.xml` and `directory-xhtml-symbol.xml` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone.

## ASP.NET Installation

1. Unzip the file called `x-web-services-asp.net.zip`. Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `directory.aspx`, `directory.config`, `directory example staff.csv`, `directory-example-product-1.txt`, `directory-example-product-2.txt`, `directory-example-product-3.txt`, `directory-example-product-4.txt`, `directory-xhtml-entities.xml`, `directory-xhtml-latin1.xml`, `directory-xhtml-special.xml` and `directory-xhtml-symbol.xml` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone. Into a sub-folder, copy the `bin` folder.

## PHP Installation

1. Unzip the file called `x-web-services-php.zip`. Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `directory.php`, `directory.config`, `directory example staff.csv`, `directory-example-product-1.txt`, `directory-example-product-2.txt`, `directory-example-product-3.txt`, `directory-example-product-4.txt`, `directory-xhtml-entities.xml`, `directory-xhtml-latin1.xml`, `directory-xhtml-special.xml` and `directory-xhtml-symbol.xml` to a folder on your Web site. Make sure this folder has the broadest possible permission settings. On Unix-based systems, set permissions to `0777`.

## Testing

In a Web browser, navigate to the URL where the Directory service is located. For example: `http://localhost/directory.asp`

If you don't see `Status: Ready` in the browser, the Web Service is not correctly installed. The most common cause of this is incorrectly set file permissions.

## Configure XStandard

To set up XStandard to use the Directory service, modify the following `<param>` tag:

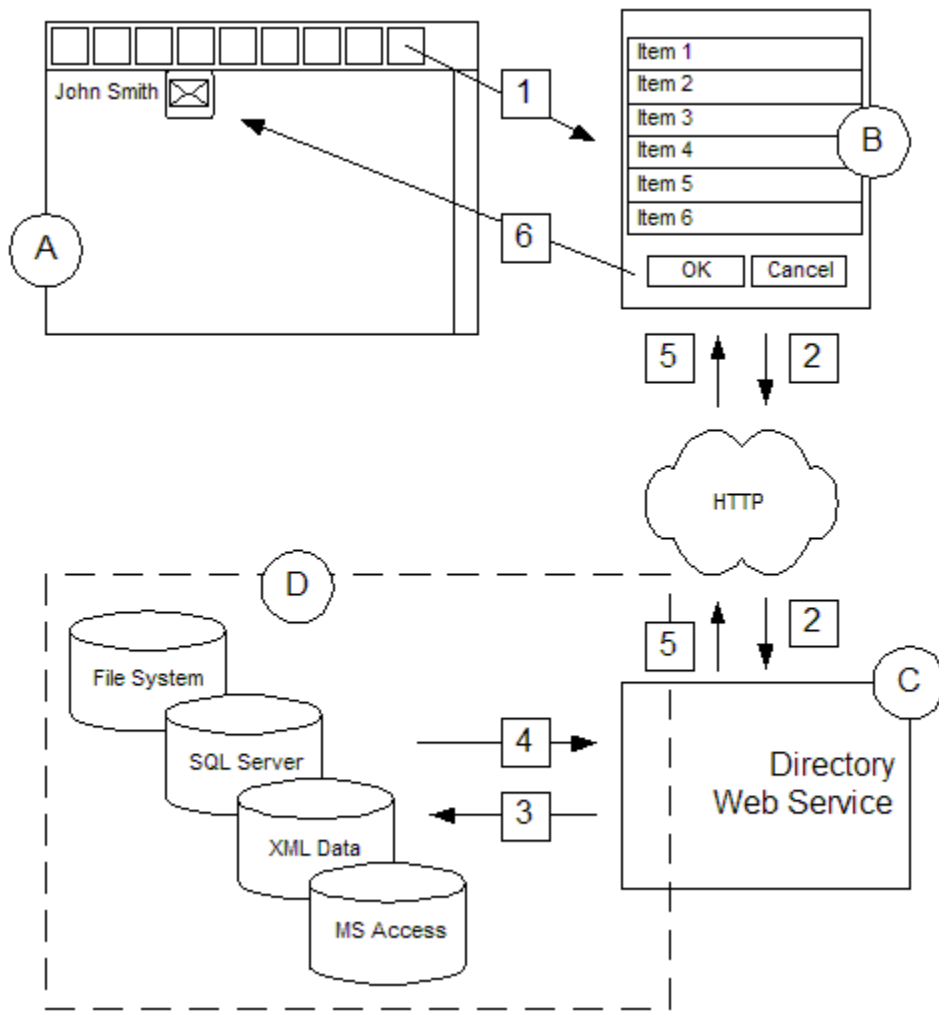
### Property DirectoryURL

(Available in XStandard Pro)

Absolute URL to a Directory Web Service. Multiple URLs can be specified, separated by a space character. For testing purposes, the following URL is available: <http://soap.xstandard.com/directory.aspx>

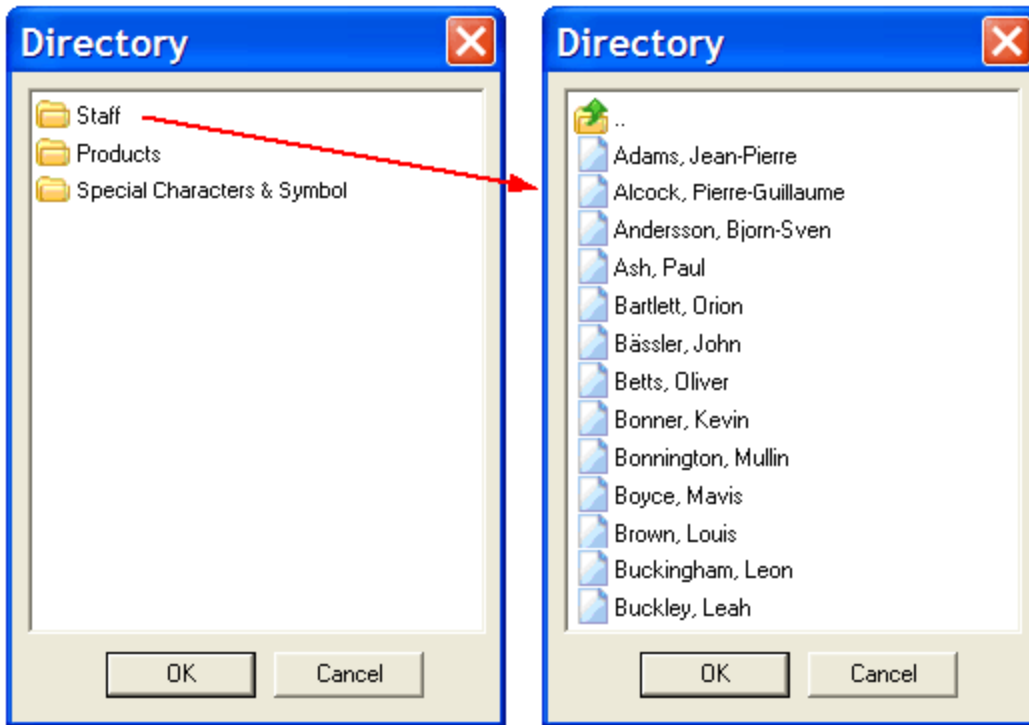
## Configure Directory Web Service

Below is a diagram illustrating how XStandard retrieves data from remote data stores using the Directory Web Service.



A user presses the Directory button on the editor's toolbar (Step 1) which brings up a dialog box (Item B). The dialog box communicates (Step 2) via HTTP with the Directory Web Service (Item C). The Web Service communicates (Step 3) with data stores (Item D) and retrieves data (Step 4). The Web Service reformats the data and sends it back to the dialog box (Step 5), populating the list box with the retrieved data. When the user selects an item from the list box, the data is inserted at the current cursor position (Step 6).

The Directory Web Service can organize data stores in a hierarchy (tree structure), permitting users to browse the data stores like folders on the file system. In the illustration below, selecting a data store displays a list of code snippets to the user.



The hierarchy of data stores and the XHTML for each snippet are constructed through a configuration file. Below is a screenshot of a sample configuration file. The file is an XML file containing a number of scripts written in VBScript or JavaScript (for the ASP version), in C# or Visual Basic .NET (for the ASP.NET version) or in PHPscript (for the PHP version). With one exception, each script is identified by a unique ID. The script without an ID is called only when the user opens the Directory dialog box for the first time. In the screenshot above, this script has created 3 folders and associated an ID with each folder. When a folder is selected, the script with the corresponding folder ID is called. Thus, in our example, when the user selects the folder "Staff" the script with the ID "a" is executed. This script retrieves a list of employees and creates an XHTML snippet for each employee.

```

<directory>
  <rule>
    <id />
    <script language="VBScript">
      <![CDATA[
        Directory.AddContainer "Staff", "a"
        Directory.AddContainer "Products", "b"
        Directory.AddContainer "Special Characters & Symbols", "c", , "entities.xml"
      ]]>
    </script>
  </rule>
  <rule>
    <id>a</id>
    <script language="VBScript">
      <![CDATA[
        Dim objConn, rsData, strName, strXHTML

        Set objConn = CreateObject("ADODB.Connection")

        objConn.Open "Driver={Microsoft Text Driver (*.txt; *.csv)};" & _
          "Dbq=" & Directory.CurrentFolder & ";Extensions=csv"

        Set rsData = objConn.Execute("SELECT * FROM directory_example_staff.csv")
        Do Until rsData.EOF
          strName = rsData("last_name").Value & ", " & rsData("first_name").Value
          strXHTML = "<a href=""staff.asp"">" & strName & "</a> & _
            & "<img src=""mail.gif"" width=""16"" height=""16"" alt="" />"

          Directory.AddObject strName, strXHTML
          rsData.MoveNext
        Loop

        Set objConn = Nothing
        Set rsData = Nothing
      ]]>
    </script>
  </rule>
  <rule>
    <id>b</id>
    <script language="VBScript">
      <![CDATA[
        Directory.AddObject "Phone", Directory.ReadFile(Directory.CurrentFolder & "\phone.txt")
        Directory.AddObject "Clock", Directory.ReadFile(Directory.CurrentFolder & "\clock.txt")
        Directory.AddObject "Radio", Directory.ReadFile(Directory.CurrentFolder & "\radio.txt")
        Directory.AddObject "Toaster", Directory.ReadFile(Directory.CurrentFolder & "\toaster.txt")
        Directory.AddObject "TV", Directory.ReadFile(Directory.CurrentFolder & "\tv.txt")
      ]]>
    </script>
  </rule>
</directory>

```

## Customization For ASP Version

The script engine has a root object called "Directory". This object is global and does not need to be instantiated. It is used to create folders and code snippets. Below is the API reference for the Directory object.

### **Sub AddContainer(*sName* As String, *sID* As String, [*sMetadata* As String], [*sLocation* As String], [*sIcon* As String])**

Create a folder. *sName* is the name of the folder. *sID* is the ID associated with the folder. *sMetadata* is additional data. *sLocation* is a URL to an alternate Directory Web Service and can be absolute or relative.

### **Sub AddObject(*sName* As String, *sValue* As String, [*sIcon* As String])**

Create a code snippet.

## Property CurrentFolder As String

Path to the folder containing the configuration file.

## Property ID As String

(read-only)

ID of the script currently being executed.

## Property Lang As String

(read-only)

Language code transmitted in the request.

## Sub LogToFile(sText As String)

Write message to log file.

## Property Metadata As String

(read-only)

Additional data.

## Function ReadFromFile(sPath As String) As String

Read contents of a file.

## Function URLEncode(sText As String) As String

Escape characters for use in URLs.

## Function XHTMLEscape(sText As String) As String

XHTML escape markup. For example, change `<` to `&lt;` and `&` to `&amp;`

## Customization For ASP.NET Version

The script engine has a root object called "XStandard.Directory". This object is global and does not need to be instantiated. It is used to create folders and code snippets. Below is the API reference for the XStandard.Directory object.

### **public void AddContainer(string *name*, string *id*, string *metadata*, string *location*, string *icon*)**

Create a folder. *name* is the name of the folder. *id* is the ID associated with the folder. *metadata* is additional data. *location* is a URL to an alternate Directory Web Service and can be absolute or relative.

### **public void AddContainer(string *name*, string *id*, string *metadata*, string *location*)**

Create a folder.

### **public void AddContainer(string *name*, string *id*, string *metadata*)**

Create a folder.

### **public void AddContainer(string *name*, string *id*)**

Create a folder.

### **public void AddObject(string *name*, string *data*, string *icon*)**

Create a code snippet.

### **public void AddObject(string *name*, string *data*)**

Create a code snippet.

### **public string CurrentFolder**

Path to the folder containing the configuration file.

### **public string ID**

(read-only)

ID of the script currently being executed.



## public string Lang

(read-only)

Language code transmitted in the request.

## public void LogToFile(string message)

Write message to log file.

## public string Metadata

(read-only)

Additional data.

## public string ReadFromFile(string path)

Read contents of a file.

## public string URLEncode(string text)

Escape characters for use in URLs.

## public string XHTMLEscape(string text)

XHTML escape markup. For example, change `<` to `&lt;` and `&` to `&amp;`

## Customization For PHP Version

The script engine can access a class stored in a `$directory` variable. This variable is already initialized. The object it contains is used to create folders and code snippets. Below is the API reference for this class.

## function add\_container(\$name, \$id, \$metadata = "", \$location = "", \$icon = "")

Create a folder. `$name` is the name of the folder. `$id` is the ID associated with the folder. `$metadata` is additional data. `$location` is a URL to an alternate Directory Web Service and can be absolute or relative.

## function add\_object(\$name, \$value, \$icon = "")

Create a code snippet.

## function current\_folder()

Path to the folder containing the configuration file.

## var id

(read-only)

ID of the script currently being executed.

## function log\_to\_file(\$msg)

Write message to log file.

## var metadata

(read-only)

Additional data.

## function read\_from\_file(\$path)

Read contents of a file.

## function xhtml\_escape(\$text)

XHTML escape markup. For example, change `<` to `&lt;` and `&` to `&amp;`

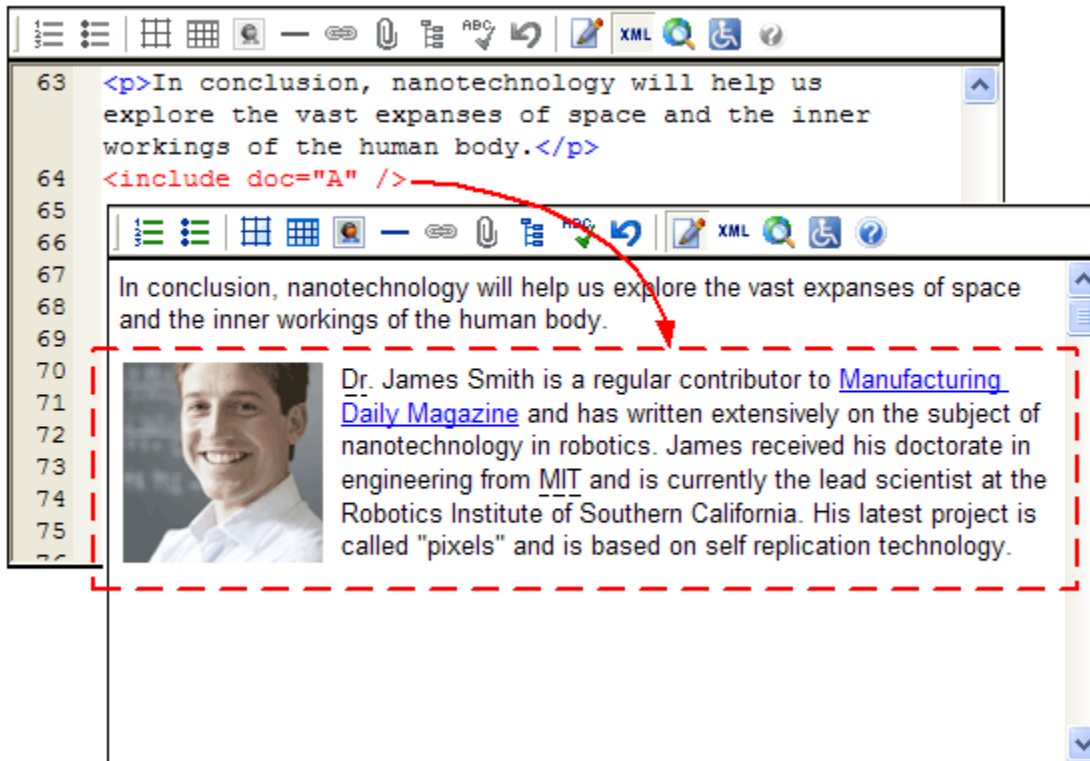
## Subdocument

- [Overview](#)
- [Requirements](#)
- [ASP Installation](#)
- [ASP.NET Installation](#)
- [PHP Installation](#)

- [Testing](#)
- [Configure XStandard](#)
- [Configure Subdocument Web Service](#)
- [Notes](#)

## Overview

Subdocuments are chunks of reusable content that authors insert into documents as required. Subdocuments are essentially custom elements that act as placeholders for content stored outside the document, within the CMS. However, whereas placeholders appear in the document as icons, in lieu of content, subdocuments display the content itself inside the document. One example of a reusable subdocument is an author's biography that can be inserted at the end of a number of articles.



The Subdocument Web Service serves subdocuments to XStandard. XStandard then replaces custom elements with content from subdocuments and renders them in WYSIWYG mode as read-only content.

## Requirements

- ASP / ASP.NET on Windows
- PHP 4.3.0+ on Linux / FreeBSD / Windows

## ASP Installation

1. Run the setup program called `x-web-services-asp.exe` Instructions for downloading this program are sent to you when you request download instructions for XStandard Pro.
2. Set "Read & Execute" file permissions on `C:\Program Files\XStandard Web Services` to Everyone.
3. Copy `subdocument.asp`, `subdocument-example-A.txt`, `subdocument-example-B.txt`, `subdocument-example-C.txt`, `subdocument-example-D.txt`, `subdocument-example-E.txt` to `C:\InetPub\wwwroot` (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone.

## ASP.NET Installation

1. Unzip the file called `x-web-services-asp.zip` Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy `subdocument.aspx`, `subdocument-example-A.txt`, `subdocument-example-B.txt`, `subdocument-example-C.txt`, `subdocument-example-D.txt`, `subdocument-example-E.txt`

E.txt to C:\InetPub\wwwroot (or another path that has an IIS virtual directory mapping) and set "Read & Write" file permissions on this folder to Everyone. Into a sub-folder, copy the bin folder.

## PHP Installation

1. Unzip the file called x-web-services-php.zip Instructions for downloading this file are sent to you when you request download instructions for XStandard Pro.
2. Copy subdocument.php, subdocument-example-A.txt, subdocument-example-B.txt, subdocument-example-C.txt, subdocument-example-D.txt, subdocument-example-E.txt to a folder on your Web site. Make sure this folder has the broadest possible permission settings. On Unix-based systems, set permissions to 0777.

## Testing

In a Web browser, navigate to the URL where the Subdocument service is located. For

example: http://localhost/subdocument.asp

If you don't see Status: Ready in the browser, the Web Service is not correctly installed. The most common cause of this is incorrectly set file permissions.

## Configure XStandard

To set up XStandard to use the Directory service, modify the following <param> tag:

### Property SubdocumentURL

(Available in XStandard Pro)

Absolute URL to a Subdocument Web Service. For testing purposes, the following URL is available:

http://soap.xstandard.com/subdocument.aspx

## Configure Subdocument Web Service

In subdocument.asp, subdocument.aspx or subdocument.php, search for "ADD CUSTOM CODE HERE". The following is an example from subdocument.aspx:

```
if(soap.Action == "doSubdocumentDescribe")
{
/*
** -----
** ADD CUSTOM CODE HERE
** -----
*/
soap.AddSubdocumentDefinition("include", "doc");
}
```

The AddSubdocumentDefinition() function is used to define custom elements that will act as subdocuments. The first argument is the custom element name. The second argument is the attribute that will contain the ID of the subdocument. Based on the previous example, the following custom element will act as a subdocument:

```
<include doc="A" />
```

The following code is used to replace custom elements with content:

```
else if(soap.Action == "doSubdocumentDownload")
{
string id = soap.GetProperty("id"); //id

/*
** -----
** ADD CUSTOM CODE HERE
** -----
*/
if (id == "A") {
soap.SetSubdocument(soap.ReadFile(Server.MapPath("subdocument-example-A.txt")));
}
}
```

The if statement is used to check the subdocument ID and then the SetSubdocument() function is used to set the subdocument content from the given subdocument. In the example above, content for the subdocument is read from a file. You can customize this code to read from your CMS.

## Notes

If subdocument content contains block elements

(`<p>`, `<table>`, `<ol>`, `<ul>`, `<dl>`, `<blockquote>`, `<h1>` to `<h6>` and `<address>`), then the custom element that is used for the subdocument should be defined as a block element. This is done via the following `<param>` tag:

```
<param name="CustomBlockElements" value="include" />
```

## Toolbar Customization

- [Styles](#)
- [Buttons](#)

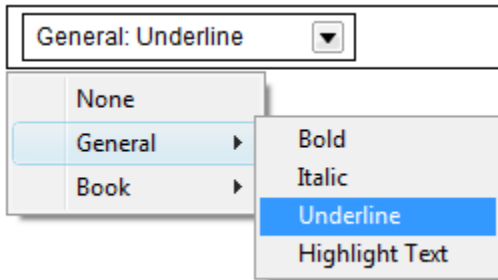
## Styles

Styles are instructions for generating markup (code). Styles are not CSS rules. CSS is often applied to the markup to create formatting. For example, a style called `Underline` may create markup that looks like this:

```
<span class="underline">Hello World</span>
```

If there is a CSS rule `span.underline {text-decoration:underline}`, then the words `Hello World` would be underlined.

The Styles seen in XStandard's drop-down menu (below) are generated by an XML document. The XML document has a simple structure and can be composed in Notepad. An example of this document can be found at "C:\Program Files\XStandard\styles.xml" on Windows, in "/Applications/XStandard/styles.xml" on OS X or [download styles.xml](#).



In order for the editor to load your styles.xml file, put styles.xml on your Web site and point the editor to this file via an absolute URL like this:

```
<param name="Styles" value="http://localhost/styles.xml" />
```

If no custom styles.xml file is specified, XStandard will use built-in styles such as `Bold`, `Italic`, `Superscript`, `Subscript`, `Heading`, `Sub-heading`, `Abbreviation`, `Computer Code`, etc.

The Styles drop-down list can be hidden from view using the following tag:

```
<param name="ShowStyles" value="no" />
```

The following XML code is an example of styles.xml file:

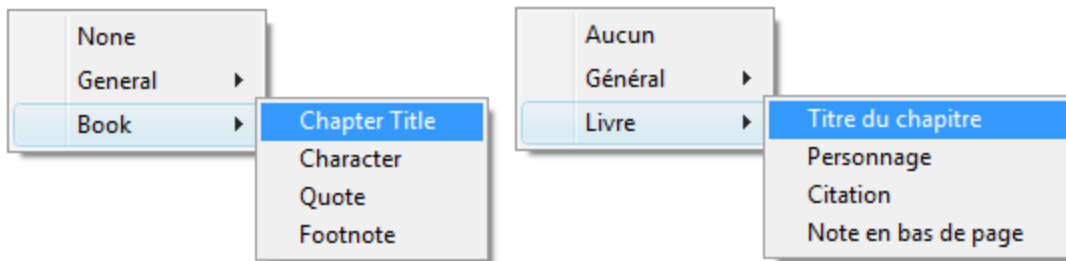
```
<styles>
<style>
<label>Bold</label>
<elt>strong</elt>
</style>
<style>
<label>Italic</label>
<elt>em</elt>
</style>
```

```

<style>
<label>Underline</label>
<elt>span</elt>
<attr>
<name>class</name>
<value>underline</value>
</attr>
</style>
<style>
<label>Highlight Text</label>
<elt>span</elt>
<attr>
<name>class</name>
<value>highlight</value>
</attr>
</style>
</styles>

```

Styles can be grouped for ease of use, as seen in the examples below. Translations for styles can be stored in the same XML document, using the "xml:lang" attribute. The `<param>` named "Lang" will determine which translation to use.



It is a good idea to get into the habit of storing XML documents in Unicode. In Notepad, select `File > Save As`, then select `Unicode` from the `Encoding` drop-down box.

The styles.xml file used to generate the bilingual example above might look like this:

```

<styles>
<group>
<label xml:lang="en">General</label>
<label xml:lang="fr">Général</label>
<style>
<label xml:lang="en">Bold</label>
<label xml:lang="fr">Caractère gras</label>
<elt>strong</elt>
</style>
<style>
<label xml:lang="en">Italic</label>
<label xml:lang="fr">Italique</label>
<elt>em</elt>
</style>
<style>
<label xml:lang="en">Underline</label>
<label xml:lang="fr">En souligné</label>
<elt>span</elt>
<attr>
<name>class</name>
<value>underline</value>
</attr>
</style>
<style>
<label xml:lang="en">Highlight Text</label>
<label xml:lang="fr">En surbrillance</label>
<elt>span</elt>

```

```

<attr>
<name>class</name>
<value>highlight</value>
</attr>
</style>
</group>
<group>
<label xml:lang="en">Book</label>
<label xml:lang="fr">Livre</label>
<style>
<label xml:lang="en">Chapter Title</label>
<label xml:lang="fr">Titre du chapitre</label>
<elt>h1</elt>
</style>
</group>
<group>
<label xml:lang="en">Character</label>
<label xml:lang="fr">Personnage</label>
<elt>character</elt>
</style>
</group>
<group>
<label xml:lang="en">Quote</label>
<label xml:lang="fr">Citation</label>
<elt>q</elt>
</style>
</group>
<group>
<label xml:lang="en">Footnote</label>
<label xml:lang="fr">Note en bas de page</label>
<elt>div</elt>
<attr>
<name>class</name>
<value>footnote</value>
</attr>
</style>
</group>
</styles>

```

This table explains each element in the styles.xml document.

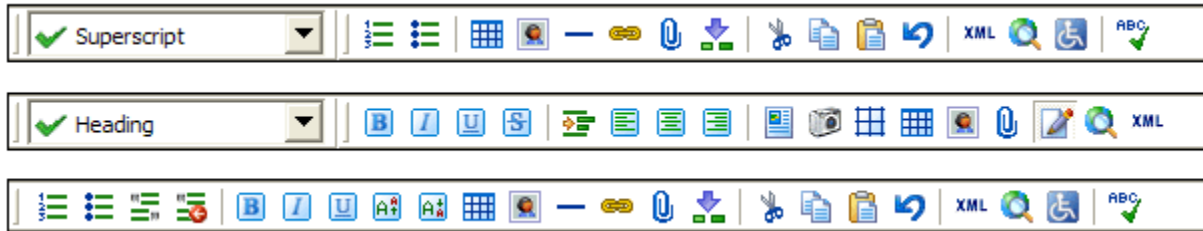
Element	Usage
<styles>	The root element that identifies this XML document as a Styles XML document. It must contain at least one <style> or <group> element.
<group>	Used to group styles together. This element must contain a <name> element and at least one <style> element. The <name> element can have an xml:lang attribute.
<style>	Used to define a style. This element must contain a <name>, <elt> and zero or more <attr> elements. The <name> element can have an xml:lang attribute. An optional <id> child element can assign a unique ID to this style.
<label>	Defines a group name or a style name.
<id>	Used to define a unique ID for a style. This ID is used in API such as ApplyStyleID() and CurrentStyles().
<elt>	This is the name of the XHTML tag to be created by the style. It must conform to XML naming rules (no spaces in the name and cannot start with a number), for example: h1 or strong.
<attr>	Used to define an attribute. It must contain one <name> element and one <value> element. The child <name> element cannot have an xml:lang attribute.
<name>	Defines an attribute name.

Element	Usage
<value>	Value of an attribute. This could be text or a function like: <code>id()</code> , <code>now()</code> , <code>date()</code> , <code>time()</code> , <code>day()</code> , <code>month()</code> , <code>year()</code> , <code>week()</code> , <code>day-of-year()</code> , <code>weekday()</code> , <code>guid()</code> , <code>random()</code> .

You can use styles to create any element with any number of attributes. See [Best Practices](#) to get the most out of the Styles feature.

## Buttons

XStandard's toolbar is totally customizable. You can show/hide buttons from a [list of predefined buttons](#) or you can define your own buttons and even change their icons. The screen shot below shows XStandard toolbar with different grouping of icons.

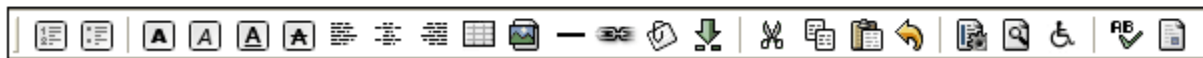


The screen shot below shows custom icons in place of XStandard default icons.

### XStandard Default Icons



### Example Of Custom Icons



Buttons on the toolbar can be used for multiple-purposes. Besides executing a predefined command such as bringing up a table or image dialog box, buttons can apply styles (similar to the styles found in the Styles drop-down list), insert code snippets or just generate an event that the parent application can hook into.

The buttons are defined in an XML document. An example of this document can be found at "C:\Program Files\XStandard\buttons.xml" on Windows, in "/Applications/XStandard/buttons.xml" on OS X or [download buttons.xml](#). If no custom buttons XML file is specified, XStandard will use built-in buttons. The list of buttons to display on the toolbar, from the list of buttons defined in the XML document, is set through the following property:

```
<param name="ToolbarWysiwyg" value="ordered-list, unordered-list, definition-list,, draw-
layout-table, draw-data-table, image, separator, hyperlink, attachment, directory,
spellchecker,, wysiwyg, source, preview, screen-reader, help" />
```

The XML for button definition looks like this:

```

<buttons>
  <cmd>
    <id>bullets</id>
    <name xml:lang="en">Bullets</name>
    <name xml:lang="fr">Liste à puces</name>
    <icon>bullets</icon>
  </cmd>
  <button>
    <id>save</id>
    <name xml:lang="en">Save</name>
    <name xml:lang="fr">Enregistrer</name>
    <toggle>no</toggle>
    <icon>save</icon>
  </button>
  <style>
    <id>underline</id>
    <name xml:lang="en">Underline</name>
    <name xml:lang="fr">Souligner</name>
    <icon>47494638396110001000E60100FFFFFFFFF00FFFBF...</icon>
    <elt>span</elt>
    <attr>
      <name>class</name>
      <value>underline</value>
    </attr>
  </style>
  <snippet>
    <id>textbox</id>
    <name xml:lang="en">Text Box</name>
    <name xml:lang="fr">Zone de texte</name>
    <icon>47494638396110001000E64500B5E2FFE2F2FBCEE...</icon>
    <value><div class="textbox"><h5>{heading}</h5><p>{text}</p></div></value>
  </snippet>
  <window>
    <id>help</id>
    <name xml:lang="en">Help</name>
    <name xml:lang="fr">Aide</name>
    <url>http://xstandard.com/help/</url>
    <icon>help</icon>
  </window>
</buttons>

```

The following tables explain each element in the buttons XML document.

XML Structure For A Button That Executes A Predefined Command	
Element	Usage
<buttons>	The root element that identifies this XML document as a buttons XML document.
<cmd>	Used to define a command button. This element must contain an <id>, <name> and <icon>. The <name> element can have an xml:lang attribute.
<id>	ID of the button.
<name>	Tooltip for the button.
<icon>	ID of an icon defined in icons.xml.
XML Structure For A Generic Button	
Element	Usage
<buttons>	The root element that identifies this XML document as a buttons XML document.
<button>	Used to define a generic button. This element must contain an <id>, <name>, <toggle> and <icon>. The <name> element can have an xml:lang attribute.



### XML Structure For A Generic Button

Element	Usage
<code>&lt;id&gt;</code>	ID of the button.
<code>&lt;name&gt;</code>	Tooltip for the button.
<code>&lt;icon&gt;</code>	ID of an icon defined in icons.xml.
<code>&lt;toggle&gt;</code>	Enables "on/off" switch behavior to the button. When set to <code>yes</code> , the button will remain depressed when clicked until will be released when clicked again. When set to <code>no</code> , the button will behave like a regular push button.

### XML Structure For A Button That Applies A Style

Element	Usage
<code>&lt;buttons&gt;</code>	The root element that identifies this XML document as a buttons XML document.
<code>&lt;style&gt;</code>	Used to define a style. This element must contain an <code>&lt;id&gt;</code> , <code>&lt;name&gt;</code> , <code>&lt;icon&gt;</code> , <code>&lt;elt&gt;</code> and zero or more <code>&lt;attr&gt;</code> elements. The <code>&lt;name&gt;</code> element can have an <code>xml:lang</code> attribute.
<code>&lt;id&gt;</code>	ID of the button.
<code>&lt;name&gt;</code>	Tooltip for the button.
<code>&lt;icon&gt;</code>	ID of an icon defined in icons.xml.
<code>&lt;elt&gt;</code>	This is the name of the XHTML tag to be created by the style. It must conform to XML naming rules (no spaces in the name and cannot start with a number), for example: <code>h1</code> or <code>strong</code> .
<code>&lt;attr&gt;</code>	Used to define an attribute. It must contain one <code>&lt;name&gt;</code> element and one <code>&lt;value&gt;</code> element. The child <code>&lt;name&gt;</code> element cannot have an <code>xml:lang</code> attribute.
<code>&lt;value&gt;</code>	Value of an attribute. This could be text or a function like: <code>id()</code> , <code>now()</code> , <code>date()</code> , <code>time()</code> , <code>day()</code> , <code>month()</code> , <code>year()</code> , <code>week()</code> , <code>day-of-year()</code> , <code>weekday()</code> , <code>guid()</code> , <code>random()</code>

### XML Structure For A Button That Inserts A Code Snippet

Element	Usage
<code>&lt;buttons&gt;</code>	The root element that identifies this XML document as a buttons XML document.
<code>&lt;snippet&gt;</code>	Used to define a button for inserting a code snippet. This element must contain an <code>&lt;id&gt;</code> , <code>&lt;name&gt;</code> , <code>&lt;value&gt;</code> and <code>&lt;icon&gt;</code> . The <code>&lt;name&gt;</code> element can have an <code>xml:lang</code> attribute.
<code>&lt;id&gt;</code>	ID of the button.
<code>&lt;name&gt;</code>	Tooltip for the button.
<code>&lt;icon&gt;</code>	ID of an icon defined in icons.xml.
<code>&lt;value&gt;</code>	XHTML to insert into the editor. The markup contained in this tag must be escaped (replace <code>&lt;</code> with <code>&amp;lt;</code> , replace <code>&gt;</code> with <code>&amp;gt;</code> , replace <code>"</code> with <code>&amp;quot;</code> and replace <code>&amp;</code> with <code>&amp;amp;</code> ).

### XML Structure For A Button That Opens A Browser Window

Element	Usage
<code>&lt;buttons&gt;</code>	The root element that identifies this XML document as a buttons XML document.
<code>&lt;window&gt;</code>	Used to define a button for opening a browser window. This element must contain an <code>&lt;id&gt;</code> , <code>&lt;name&gt;</code> , <code>&lt;url&gt;</code> and <code>&lt;icon&gt;</code> . The <code>&lt;name&gt;</code> element can have an <code>xml:lang</code> attribute.
<code>&lt;id&gt;</code>	ID of the button.
<code>&lt;name&gt;</code>	Tooltip for the button.
<code>&lt;icon&gt;</code>	ID of an icon defined in icons.xml.
<code>&lt;url&gt;</code>	An absolute URL to open in a new browser window.

[icons.xml](#) contains graphics for each icon. These graphics are 16x16 and 24x24 GIF files have have been encoded into HEX. An online tool is available to **encode GIF files into HEX**. The current release of XStandard only uses 16x16 graphics for icons so you can omit the 24x24 graphics.

## Best Practices Styles

The Styles selector creates markup (elements and attributes). Some of the markup will reference CSS in order to format data, but there is no fixed relationship between a style and formatting. In fact, data can easily be reformatted simply by changing to another CSS. Because of this, it is best to give Styles meaningful names that are related to the contents of the document, rather than names that describe formatting. So avoid names that have colors or font styles in them like "Red text" or "Arial 12 pt". Instead, use semantic-rich names like "Product Name", "Product Specs" and "Price".

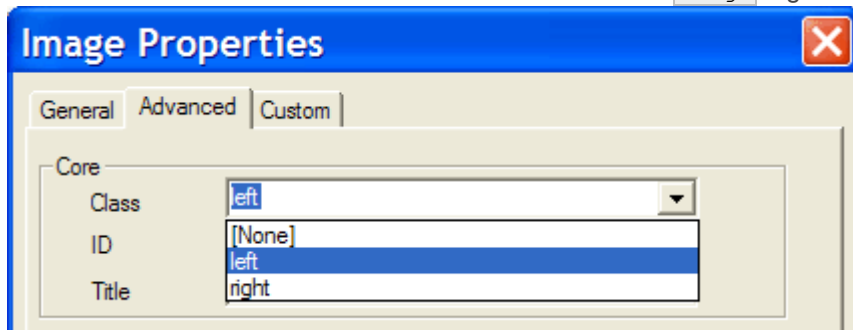
## CSS

CSS documents are made up of formatting rules. Each rule has a selector that is used to bind the rule to specific tags.

Many developers write selectors like this: `.classname {property: value}`

The dot class name syntax is a shorthand for `*.classname {property: value}`. This says that this class name can be applied to any tag. This syntax is perfectly fine if the rule truly can be applied to any tag. If not, it is better to specify the tag that the rule does apply to. For example: `img.right {float: right}`

XStandard can then use the selector to show only relevant CSS rules when editing any given tag. For example, the screenshot below shows the available CSS rules for the `<img>` tag.



## XStandard And Desktop Applications

When using XStandard in a Web environment, store the Styles, CSS, and License files on Web servers. For example:

```
<param name="Styles" value="http://server/styles.xml" />
```

When using XStandard in a desktop application, store these documents in resource files in your application project. Then pass the data from the files directly to the editor's properties. In Visual Basic, this might look like:

```
XHTMLEditor1.Styles = StrConv(LoadResData(101, "CUSTOM"), vbUnicode)
```

## Advanced Topics

- [Caching](#)
- [Heartbeat](#)
- [Placeholders](#)
- [Browser Preview Customization](#)
- [Screen Reader Preview Customization](#)
- [Namespaces](#)
- [Locking](#)

- [Markers](#)

## Caching

If you have configured the editor to reference external files, when the editor starts up, it downloads these files over the network. Since they are quite small, downloading a CSS, Styles or License file is no more resource intensive than loading a typical Web page into a Web browser. However, larger files such as a custom localization file or a custom buttons definition file can slow down the editor's start-up. To address this issue, files that need to be fetched over the network can be cached.

To enable caching, set the following param tag:

```
<param name="EnableCache" value="yes" />
```

When the editor fetches files referenced by the following param tags, the files are stored in a cache:

- CSS
- EditorCSS
- Styles
- Localization
- ScreenReaderXSLT
- PreviewXSLT
- Buttons
- Icons
- Placeholders
- License

The editor uniquely identifies each file based on a case-sensitive URL to the file. When caching is enabled, at start-up the editor checks to see if a file with a given URL is already in the cache. If it is, the editor uses the cached version of the file instead of downloading the file again from the server. The editor automatically ensures that the cache does not exceed 200 files or 5 MB.

Caching does present a challenge however. When you update a file on the server, how do you tell the editor to use the new file instead of the old file kept in the cache? The easiest and most reliable way to do this is to modify the file's URL. For example, let's say the buttons.xml file is found at the URL:

```
http://server1/buttons.xml
```

When the file is updated, modify the file's URL by appending a version number in a query string. For example:

```
http://server1/buttons.xml?14
```

Each time you modify the file, simply increment the version number. For example:

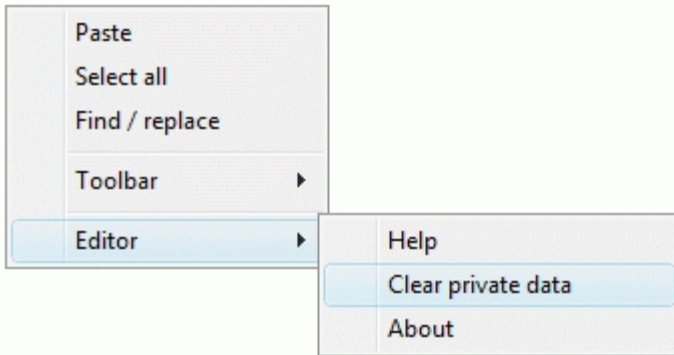
```
http://server1/buttons.xml?15
```

To automate the URL updating process, when your scripts are generating the param tags, a date last modified can be retrieved from the file and appended to the URL. For example:

12/3/2004 9:54:44 PM which when URL encoded looks like:

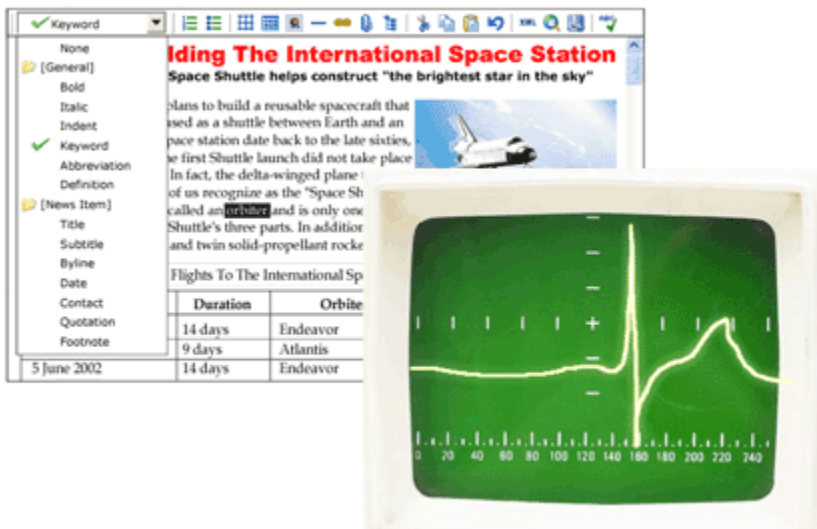
```
http://server1/buttons.xml?12%2F3%2F2004+9%3A54%3A44+PM
```

The editor's cache can be cleared manually by selecting `Editor > Clear private data` from the context menu.



## Heartbeat

Web-based content management systems log out users (i.e. lose Session state) after a period of inactivity between the browser and the Web server. Even if a browser is open and a user is authoring content, in Web terms this is considered inactivity. When Session state is lost, users are forcibly logged out and valuable work is often lost. To avoid this, XStandard Pro has a feature called "Heartbeat" which can send HTTP pulses to the server at regular intervals. The continuing stream of pulses tells the Web server that the user is still connected and that Session state should not be lost. This ensures authors remain logged into the content management system for as long as they need to complete their work.



To enable the heartbeat, create a Web page in the same development environment that your CMS is written in (ASP, PHP, etc.) and put the Web page in the same folder with the rest of your CMS Web pages. Add a line of code to the Web page in order to trigger the scripting engine. This can be as simple as writing out the current date. Here is an ASP example:

```
<%
Response.Write Now()
%>
```

Then, in the `<object>` tag for the editor, add the following `<param>` tag and point it to the location of the newly created Web page:

```
<param name="HeartbeatURL" value="http://myserver/heartbeat.asp" />
```

To set the length of the pulse interval, use the following `<param>` tag where the value is measured in seconds:

```
<param name="HeartbeatInterval" value="300" />
```

# Placeholders


## Overview

Placeholders are empty custom tags that reserve space for dynamic content that is inserted when a Web page is requested. For example, the markup:

```
Today's temperature is <temperature />.
```

...gives this result when the placeholder is replaced by the current temperate:

```
Today's temperature is 23°C.
```

In the editor's WYSIWYG mode, placeholders display by default as  icons. If the placeholder has a `title` attribute, its value will display as a tooltip when the cursor is placed over the icon.

Custom icons such as the one seen below can be assigned or mapped to placeholders via the placeholders.xml file.





The placeholders.xml file is referenced via the "Placeholders" `<param>` tag. For example:

```
<param name="Placeholders" value="http://yourserver/placeholders.xml" />
```

The following is an example of a placeholders.xml file that maps to a custom icon:

```
<placeholders>
<placeholder>
<elt>temperature</elt>
<icon>thermometer</icon>
</placeholder>
</placeholders>
```

In the example above, the `<placeholders>` tag is a root element and contains one or more `<placeholder>` elements. The `<placeholder>` element defines the rules for matching an icon to a custom tag. The `<elt>` tag contains the name of the custom tag. The `<icon>` element contains the ID of an icon defined in the icons.xml file, or it can contain the [HEX value of a GIF](#) file.

You can assign icons to placeholders based on the element name combined with the value of its attributes. For example, you might want the placeholder  to display in the editor when the `type` attribute value is "happy", and the  placeholder to display when the `type` attribute value is "sad":


```
<emoticon type="some value" />
```

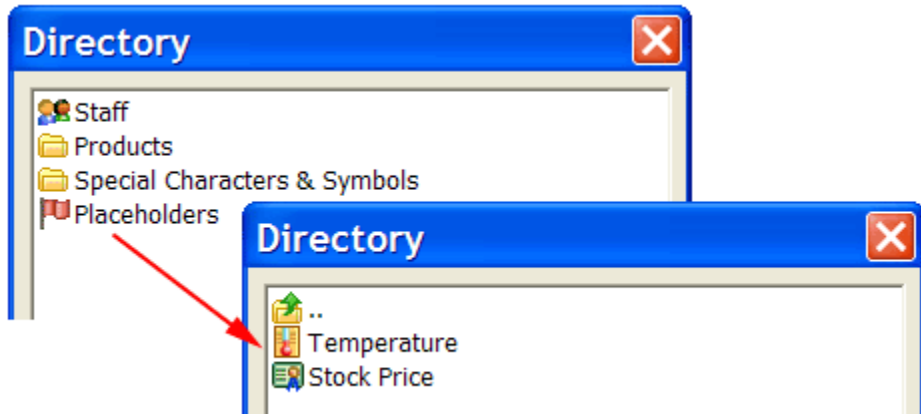
In the example below, use the `<attr>` element within the `<placeholder>` element to define the attribute to match on. The `<name>` element is the name of the attribute and the `<value>` element contains the value of the attribute. For example:

```
<placeholders>
<placeholder>
<elt>emoticon</elt>
<attr>
<name>type</name>
<value>happy</value>
</attr>
<icon>emoticon-happy</icon>
</placeholder>
<placeholder>
<elt>emoticon</elt>
<attr>
<name>type</name>
<value>sad</value>
```

```
</attr>
<icon>emoticon-sad</icon>
</placeholder>
</placeholders>
```

## Inserting Placeholders

The easiest way for non-technical users to insert placeholders is through the Directory service (  on the toolbar). The screenshot below shows the user browsing the Placeholders folder for appropriate placeholders.



## Configuring XStandard To Preview Dynamic Content

The editor's Browser Preview mode is the result of taking content entered into the editor and running it through an XSLT. A copy of this XSLT file can be found in "C:\Program Files\XStandard\preview.xml" on Windows and in "/Applications/XStandard/preview.xml" on OS X. The editor references this file via the "PreviewXSLT" `<param>` tag. For example:

```
<param name="PreviewXSLT" value="http://myserver/preview.xml" />
```

The XSLT can be customized to replace placeholders with dynamically generated content, so that the results can be seen in Browser Preview mode. For example, say we want to implement a document include capability in the editor. The following placeholder can be used to identify the document fragment to be included:

```
<include document="ABC123" />
```

In order to display the contents of the included document fragment in Browser Preview mode, we can add the following rule to the XSLT:

```
<xsl:template match="include">
<xsl:variable name="url">http://yourserver/document.asp?id=<xsl:value-of
select="@document"/></xsl:variable>
<xsl:apply-templates select="document($url)/*" />
</xsl:template>
```

This rule states that when the `<include>` element is encountered, take the value from the `document` attribute and pass it in the query string to URL `http://yourserver/document.asp?id=xxx`. Then display the data returned from this URL. Note, the data returned must be a valid XHTML (XML) document fragment with a root element. Use a `<span>` tag as a root element if the contents are all inline elements. For example:

```
<span>The stock price of <abbr>IBM</abbr> is $123.15.</span>
```

Use a `<div>` tag as a root element if the contents contain a block element. For example:

```
<div><p>...</p><table>...</table><p>...</p></div>
```

Since placeholders are treated as inline elements by the editor, a placeholder will likely be inside a `<p>` tag. Add the following rule to the XSLT that will convert the `<p>` tag that contains a placeholder, such as `<include>`, into a `<div>` tag.

```
<xsl:template match="p[include]">
<div>
<xsl:apply-templates/>
```

```
</div>
</xsl:template>
```

## Browser Preview Customization

XStandard is a content editor, therefore content authors use the editor to edit portions of a Web page, not the entire page. Some authors find it useful however to preview the content they produce within the layout of the actual Web page. You can add this feature by modifying the XSLT in preview.xsl, in order to customize the Browser Preview mode.

To instruct the Browser Preview feature to use a different CSS file, add the following XSLT rule to preview.xsl:

```
<xsl:template match="style">
<style type="text/css" media="screen">@import url('http://myserver/format.css');</style>
</xsl:template>
```

Layout markup can be added to preview.xsl in the following way:

```
<xsl:template match="body">
<body>
<h1></h1>
<ul id="nav">
<li><a href="#" onclick="return false;">Home</a></li>
<li><a href="#" onclick="return false;">Products</a></li>
<li><a href="#" onclick="return false;">Services</a></li>
<li><a href="#" onclick="return false;">News</a></li>
<li><a href="#" onclick="return false;">About Us</a></li>
</ul>
<div id="content">
<b>xsl:apply-templates />
</div>
</body>
</xsl:template>
```

Content generated through the editor will be inserted into the page layout markup at the location `<xsl:apply-templates />`.

To instruct the editor to use the customized Browser Preview file, give the location of the file in the `<param>` tag named `PreviewXSLT`. For example:

```
<param name="PreviewXSLT" value="http://myserver/preview.xsl" />
```

## Screen Reader Preview Customization

The Screen Reader Preview feature, which can be customized or completely replaced by a specialized version, is written in XSLT. XSLT is a language for transforming (restructuring) XML documents. Since XHTML is an XML language, converting it to another structure is easy using XSLT.

To instruct the editor to use a custom screen reader preview file, give the location of the file in the `<param>` tag named `ScreenReaderXSLT`. For example:

```
<param name="ScreenReaderXSLT" value="http://myserver/screenreader.xsl" />
```

## Namespaces

Namespaces are a way to distinguish tag names for different XML vocabularies. For example, if bookstore A and bookstore B use the tag name `<book>` to identify a book, there can be a conflict when data from both bookstores is used in the same document.

To avoid the conflict, each bookstore's tags can be grouped into a namespace. Namespaces are unique names, usually written in the form of a URL, that identify a set of tags. Thus, bookstore A can define a namespace such as `http://apple-books` and bookstore B can define a namespace such as `http://big-books`.

Since namespaces are quite long, repeating the entire namespace for each tag can be quite bulky. Therefore, a short name called a prefix is associated with each namespace, and the association made in the markup like this:

```
<p xmlns:a="http://apple-books" xmlns:b="http://big-books"> ... </p>
```

Alternatively, the association can be specified in the Namespaces `<param>` tag. For example:

```
<param name="Namespaces" value="xmlns:a='http://apple-books' xmlns:b='http://big-books'" />
```

Once namespaces are declared and the association established between a prefix and a namespace, the prefix can be used to make each `<book>` tag unique. For example:

```
<p>
Buy <a:book>Easy XHTML</a:book> from Apple Books<br />
Buy <b:book>Easy XHTML</b:book> from Big Books
</p>
```

## Locking

In WYSIWYG mode, areas of content can be locked (made read-only). Elements can be locked by specifying the following CSS vendor-specific extension:

Name	Values	Initial value	Applies to	Inherited?	Percentages	Media groups
<code>-xs-lock</code>	yes   no   true   false   inherit	no	all	yes	n/a	all

The following example locks all elements except an element with ID "content":

```
body {
color: black;
background-color: white;
-xs-lock: yes;
}
#content {
border: 1px dashed red;
-xs-lock: no;
}
```

The `-xs-lock` property can be added to a CSS file that is referenced by the following `<param>` tag:

```
<param name="CSS" value="http://yourserver/format.css" />
```

Or it can be added to an editor specific CSS file referenced by the following `<param>` tag:

```
<param name="EditorCSS" value="http://yourserver/editor.css" />
```

Locking properties can also be applied at run-time. Here is a Visual Basic example:

```
XHTMLEditor1.EditorCSS = "h1 {-xs-lock: yes}"
```

## Markers

In WYSIWYG mode, areas of content can be flagged with markers. Markers are labels that denote the start and end of a specific element. Markers are created using the following CSS vendor-specific extensions:



Name	Values	Initial value	Applies to	Inherited?	Percentages	Media groups
-xs-marker-label	element-name   attr(<identifier>)   <string>	element-name	h1, h2, h3, h4, h5, h6, p, blockquote, table, address, div, ul, ol, dl, a, abbr, acronym, cite, code, dfn, em, kbd, samp, strong, var, big, small, sub, sup, tt, img, object, label, q, span, [custom elements]	no	n/a	all
-xs-marker-color	<color>	black	h1, h2, h3, h4, h5, h6, p, blockquote, table, address, div, ul, ol, dl, a, abbr, acronym, cite, code, dfn, em, kbd, samp, strong, var, big, small, sub, sup, tt, img, object, label, q, span, [custom elements]	no	n/a	screen
-xs-marker-background-color	<color>	#ffff99	h1, h2, h3, h4, h5, h6, p, blockquote, table, address, div, ul, ol, dl, a, abbr, acronym, cite, code, dfn, em, kbd, samp, strong, var, big, small, sub, sup, tt, img, object, label, q, span, [custom elements]	no	n/a	screen
-xs-marker-border-color	<color>	black	h1, h2, h3, h4, h5, h6, p, blockquote, table, address, div, ul, ol, dl, a, abbr, acronym, cite, code, dfn, em, kbd, samp, strong, var, big, small, sub, sup, tt, img, object, label, q, span, [custom elements]	no	n/a	screen

The following example displays a marker around an element with ID "content":

```
#content {
  -xs-marker-label: "Press release body text";
}
```

The following example will display a marker around a custom element <price> and use the data in the title attribute for a label.

```
price {
  -xs-marker-label: attr(title);
}
```

The following example displays a red color marker around a locked element:

```
#footer {
  -xs-marker-label: "Read-only: do not edit";
  -xs-marker-color: white;
  -xs-marker-background-color: red;
  -xs-lock: yes;
}
```

The marker CSS properties can be added to a CSS file that is referenced by the following <param> tag:

```
<param name="CSS" value="http://yourserver/format.css" />
```

Or they can be added to an editor specific CSS file referenced by the following <param> tag:

```
<param name="EditorCSS" value="http://yourserver/editor.css" />
```

Marker properties can also be applied at run-time. Here is a Visual Basic example:

```
XHTMLEditor1.EditorCSS = "h1 {-xs-marker-label: 'Heading'}"
```

## Note

By default, content within markers is displayed inside a dashed outline. To remove the dashed outline, set the CSS border to "none". For example:

```
#content {
```

```
-xs-marker-label: "Press release body text";  
border: none;  
}
```

XStandard removes inline elements that have no content. For example, given the following markup:

```
<p>  
Text text <span></span> text text.  
</p>
```

The element `<span>` will be removed because it has no content. If there is a marker on the empty `<span>` element, the marker will be removed as well. If you require markers on inline content, use custom elements like this:

```
<p>  
Text text <first-name></first-name> text text.  
</p>
```

## License File

A license file transforms XStandard Lite into Pro. If you are using XStandard Lite, you do not need any license files. If you are using a 30-day evaluation version of XStandard Pro, your license file can be found at "C:\Program Files\XStandard\license.txt" on Windows or "/Applications/XStandard/license.txt" on OS X. The license is good for 30 days and permits users of the editor to upload files, browse file libraries and spell check content using Web Services located on the XStandard server or on localhost. If, for evaluation purposes, you require a different URL to locate your Web Services, please [contact us](#).

Specify the location of a license file in the `<param>` tag called `License`. Use an absolute URL like this:

```
<param name="License" value="http://myserver/license.txt" />
```

## Did You Know?

- [FAQs](#)
  - [Did you know that if you can create line breaks with a keyboard shortcut?](#)
  - [Did you know that you can put a cursor in front of and behind block elements?](#)
  - [Did you know that you can use custom tags as placeholders for dynamic content?](#)
  - [Did you know that the output code from XStandard is an XML fragment?](#)
  - [Did you know that you can omit mailto: when creating a hyperlink to an email address?](#)
  - [Did you know that you can create a multi-row toolbar?](#)
  - [Did you know that you can add image alignment options to the image context pop-up menu?](#)

## FAQs

### *Did you know that if you can create line breaks with a keyboard shortcut?*

Press `Shift-Enter`, you get a `<br />`.

### *Did you know that you can put a cursor in front of and behind block elements?*

Block elements are `<div>`, `<table>`, `<ol>`, `<ul>`, `<blockquote>` and `<hr>`. See this by pressing the left or right arrow keys until the cursor displays alongside content, as shown in the screenshot below. This is very useful when you have 2 `<div>` or `<table>` structures next to each other and you need to add text or objects between them.

Cups of coffee consumed by each person			
Name	Cups	Type	Sugar
Wendy	10	Regular	yes
Jim	15	Decaf	no

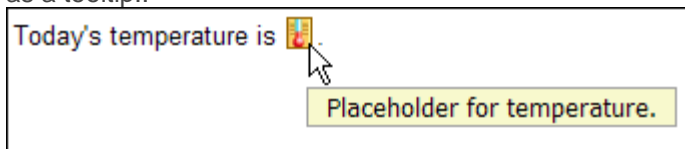
**Did you know that you can use custom tags as placeholders for dynamic content? Create markup like this:**

```
<p>
Today's temperature is <temperature title="Placeholder for temperature."/>.
</p>
```

... and when you display the content on a Web page, the custom tag (placeholder) is replaced by dynamic data. The markup above gives the following result when the placeholder is for the current temperature:

Today's temperature is 23°C.

As seen in the screenshot below, you can make custom tags easier to recognize and apply by assigning them custom icons and tooltips, using the `placeholders.xml` file. The text used in the `title` attribute for the custom tag displays as a tooltip..



**Did you know that the output code from XStandard is an XML fragment?**

This means that you can load it into DOM XML parser or process it with XSLT.

Since XStandard is a content editor, the markup it generates is an XML fragment without a root element. So, before you load this markup into an XML parser, you need to add the root element yourself. Here is a Visual Basic 6 example:

```
Dim objDoc As MSXML2.DOMDocument40
Set objDoc = New MSXML2.DOMDocument40
objDoc.async = False
objDoc.loadXML "<root>" & XHTMLEditor1.Value & "</root>"
MsgBox objDoc.xml
Set objDoc = Nothing
```

Here is the same example in C#:

```
XmlDocument doc = new XmlDocument();
XmlNamespaceManager namespaceManager = new XmlNamespaceManager(doc.NameTable);
doc.LoadXml("<root>" + axXHTMLEditor1.Value + "</root>");
MessageBox.Show(doc.InnerXml.ToString());
```

**Did you know that you can omit `mailto:` when creating a hyperlink to an email address?**

XStandard automatically inserts `mailto:` for you.

**Did you know that you can create a multi-row toolbar?**

Insert a `;` between button IDs in the `Toolbar` param tag. For example:

```
<param name="ToolbarWysiwyg" value="ordered-list,unordered-list;image,hyperlink" />
```

**Did you know that you can add image alignment options to the image context pop-up menu?**

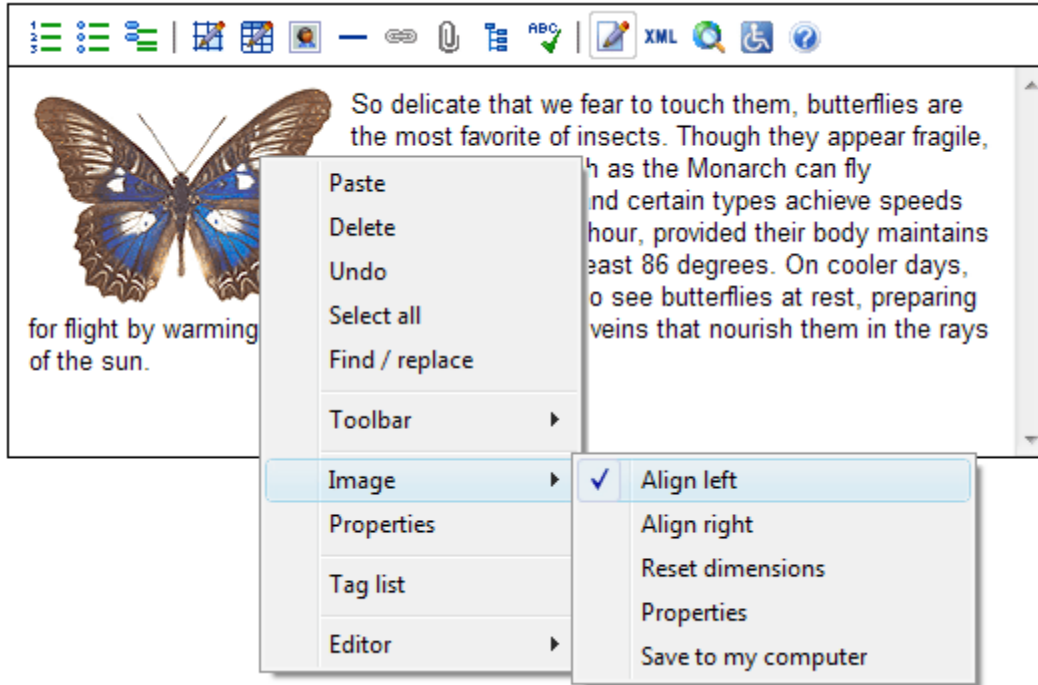
Create CSS classes for image alignment like this:

```
img.left {float:left}
img.right {float:right}
```

Then let the editor know about these CSS classes like this:

```
<param name="ClassImageFloatLeft" value="left" />
<param name="ClassImageFloatRight" value="right" />
```

The screenshot belows shows the result:



## Changes From Previous Version

### API Changes

- `ModeChange()` event has been renamed to `ModeChanged()`
- `ButtonClick()` event has been renamed to `ButtonClicked()`
- The following API have been removed:
  - `Proxy`
  - `EnableProxy`
  - `EnableProxyAuthentication`
  - `Toolbar`
  - `Expand`
- The following API have been added:
  - `ProxySetting`
  - `ProxyServer`
  - `ContextMenuActivated()`
  - `ContextMenuClicked()`
  - `ClearContextMenu()`
  - `ToolbarWysiwyg`
  - `ToolbarSource`
  - `ToolbarPreview`
  - `ToolbarScreenReader`

- [ExpandToolbarWysiwyg](#)
- [ExpandToolbarSource](#)
- [ExpandToolbarPreview](#)
- [ExpandToolbarScreenReader](#)
- [ToolbarEffect](#)

## Further Reading

- [XStandard Upgrade Guide](#)

# Copyright

Copyright © 2002-2010 Belus Technology, Inc.

Third-party licenses:

- This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>). Copyright (c) 1998-2005 The OpenSSL Project. This product includes cryptographic software written by Eric Young ([eyay@cryptsoft.com](mailto:eyay@cryptsoft.com)). This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com)).
- Copyright © 1998-2003 [Daniel Veillard](#)
- Copyright © 2002 [W3C](#)
- Copyright © 1995-2003 [Jean-loup Gailly and Mark Adler](#)
- Copyright © 1998-2003 [John Maddock \(Boost Software\)](#)
- Copyright © 1991-1998 [Thomas G. Lane](#)
- Copyright © 2004 [Glenn Randers-Pehrson](#)
- Copyright © 2004 [Kevin Atkinson](#)
- Copyright © 2001-2005 [Mike Krueger](#)
- Copyright © 2004 [John Heinstejn](#)
- Copyright © 2003 [Vincent Blavet](#)
- This product includes features licensed by Vlad Alexander