

REALOBJECTS

edit-on[®] Pro 3.x

Integration Manual

Manual Rev 3.1.195 – 2005-02-04



REALOBJECTS

Cross Platform Software

Copyright © 2000-2005 RealObjects GmbH, Saarbrücken, Germany.
All Rights Reserved.

edit-on® is a registered trademark of RealObjects GmbH, Saarbrücken, Germany.

WWW.REALOBJECTS.COM

Table of Contents

Table of Contents	2
1 Introduction	4
2 edit-on Pro API Reference	6
2.1 <i>Applet Parameter API</i>	6
2.2 <i>Configuration File</i>	12
2.2.1 XML Format	12
2.2.2 Sample Configuration File in XML Format	16
2.3 <i>Toolbar Definition</i>	18
2.3.1 XML Format	18
2.3.2 Functions	19
2.3.3 Custom Button	31
2.3.4 Sample Toolbar Definition File in XML Format	32
2.4 <i>JavaScript API</i>	34
2.5 <i>JavaScript API for Mac OS</i>	38
2.5.1 Introduction	38
2.5.2 Restrictions and technical constraints of the JavaScript API for Mac OS	41
2.5.3 API Description for Macintosh platform	42
2.5.4 Integration of the JavaScript API for Macintosh	48
2.5.5 How to use the JavaScript API for Macintosh when submitting a form	49
2.6 <i>Style Sheets</i>	50
2.6.1 Editable Styles and Read Only Styles	50
2.6.2 Importing Style Sheets	50
2.6.3 Exporting Style Sheets	51
2.6.4 Styles Supported by edit-on Pro	52
2.7 <i>Custom XML Tags and Unknown Tags</i>	54
2.7.1 Custom XML Tags	54
2.7.2 Unknown Tags	55
2.8 <i>Read Only Areas and Read Only Tags</i>	56
2.9 <i>Integrating the Clean Up Process</i>	56
2.9.1 The Client Side Clean Up Process	57
2.9.2 The Server Side Clean Up Process	58
2.9.3 Sample cases	59
2.9.4 Error Messages	63
2.10 <i>Custom Dialog API</i>	63
2.10.1 Introduction	63
2.10.2 Reference	64
2.10.3 Sample	77
3 Integrating edit-on Pro	79
3.1 <i>License Key Mechanism</i>	79

3.2	<i>Calling public edit-on Pro methods with JavaScript</i>	79
3.3	<i>Event Handling with JavaScript</i>	83
3.4	<i>Using direct HTTP connections</i>	86
3.5	<i>Image Root</i>	87
3.6	<i>Applet Caching Mechanism</i>	91
3.6.1	Applet Caching Mechanism using the Microsoft Java Virtual Machine (Java Package Manager)	91
3.6.2	Applet Caching Mechanism using the Sun JRE (Java Plug In Caching Mechanism)	92
4	Using edit-on Pro	95
4.1	<i>Browser Settings</i>	95
4.1.1	Setting Up Microsoft Internet Explorer with Microsoft Java VM	95
4.1.2	Setting Up Netscape 6.x, 7.x & Mozilla 1.x with Sun JRE 1.4.1	97
4.2	<i>Using the edit-on Pro Editor</i>	98
4.2.1	Hot Keys	98
4.2.2	Using The Copy/Cut/Paste functionalities	99
4.2.3	Style Sheets Support	99
4.2.4	Displaying Unknown Tags	103
4.2.5	Online Help	103
4.3	<i>Java Security Considerations</i>	103
4.3.1	Running edit-on Pro as a signed applet	103
4.3.2	Removing RealObjects from Trusted Publisher in Microsoft Internet Explorer	105
4.3.3	Removing RealObjects from Granted Vendors in Netscape Navigator	106
4.4	<i>Enabling the Spelling Checker</i>	108
4.5	<i>Spelling Checker Properties</i>	108
4.6	<i>MainLexicon and UserLexicon</i>	109
4.6.1	Creating a Custom Lexicon (Dictionary)	110
4.7	<i>Spelling Checker Options</i>	110
4.8	<i>Example Properties File</i>	112
4.9	<i>Adding Words to the Spelling Checker</i>	113
5	LOCALE Setting	115
5.1	<i>Changing the LOCALE</i>	115
5.2	<i>Custom Localization</i>	115
5.3	<i>Sample Custom Localization</i>	115
6	Troubleshooting Tips	117

1 Introduction

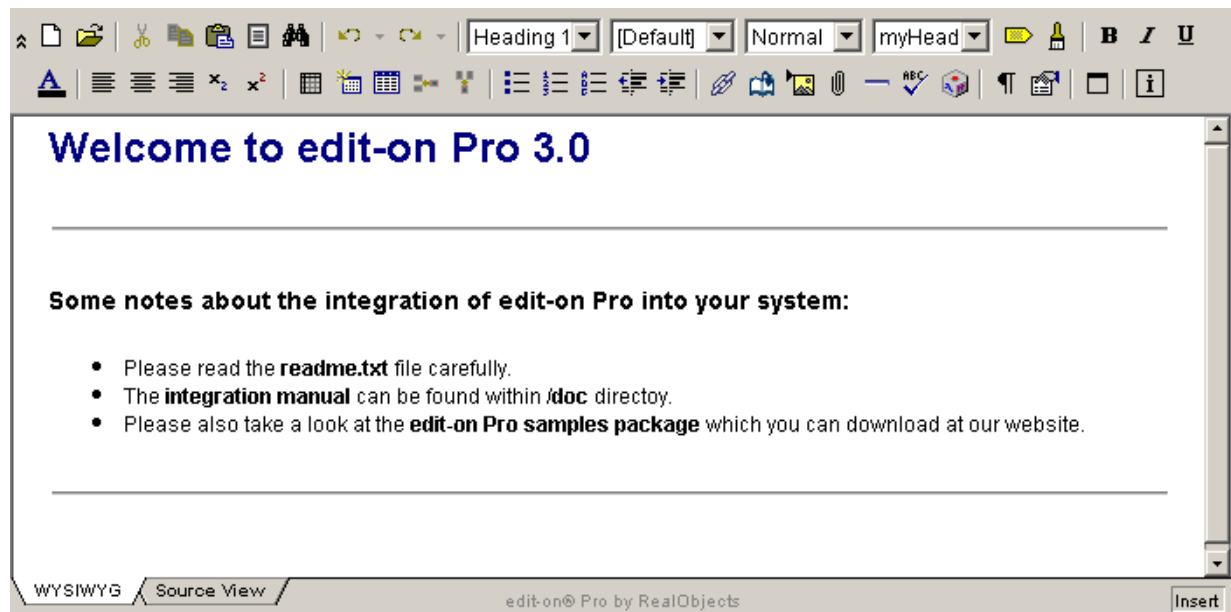
RealObjects edit-on Pro is a cross-platform, in-browser WYSIWYG editor which enables XHTML content authoring with XML markup. Developed as applet in pure Java 1.1, edit-on Pro does not require special libraries or client plugins. The editor also runs on Mac OS 9 and Mac OS X.

edit-on Pro empowers non-technical users to become content contributors without any knowledge of HTML or XML. It has an easy-to-use, intuitive user interface which provides word processor-like and XML editor-like features to online applications, including copy/paste functions from/to for example MS Word or MS Excel. This makes edit-on Pro 3.1 the most versatile multi-platform in-browser authoring tool for a wide range of Content Management Systems, e-Learning, Knowledge Management Systems, and CRM.

The comprehensive custom XML tag functionality and support for read-only elements and read-only areas make edit-on Pro the superior tool whenever in-browser XHTML/XML editing capabilities are needed. The read-only elements feature allows the definition of elements which cannot be removed, but rather edited, by the user. Read-only areas, on the other hand, feature a capability to restrict the user both from removing and editing a certain part of the document.

The editor outputs well-formed XHTML which can be easily parsed and automatically transformed using XSLT. An internal XML parser integrated within edit-on Pro always ensures the wellformedness of the document. The XHTML native format of edit-on Pro requires that only wellformed documents can be loaded into the editor, that is, when the need for a clean-up process arises. A client-side or server-side clean up process (TIDY from W3C) can be integrated into edit-on Pro to help to ensure the wellformedness of the imported document. The integrator can further specify the external clean-up process using a custom client-side or server-side application.

edit-on Pro also features basic Cascading Style Sheets (CSS) support, which allows the editor to import, display, edit, and export basic cascading style sheets. The use of cascading style sheets provides an ideal way to display certain parts of the document in accordance to W3C standards. CSS was introduced in HTML 4.0 and it is very useful for separating the content and presentation layout of documents.



For seamless integration within applications and platforms, edit-on Pro offers a variety of parameters, APIs and configuration possibilities, which are described in this manual. The basic integration of edit-on Pro into a web page using the <applet> tag looks like this:

....

```
<APPLET code="com.realobjects.eop.applet.EditorApplet" archive="edit-on-pro-signed.jar"
height=400 width=750 name=HTMLEditor MAYSCRIPT>
    <PARAM name="CODEBASE" value="http://www.yourdomain.com/eopro/">
    <PARAM name="CABBASE" value="edit-on-pro-signed.cab">
    .
    .
</APPLET>
....
```

Note: Changes within the blue/italic marked part of the APPLETTAG are optional, while the CODEBASE path (green/bold) has to be specified to match your needs. The CODEBASE URL must point to the directory in which the edit-on Pro CAB/JAR/class/icon files are installed. The MAYSCRIPT statement makes the applet able to call JavaScript functions via the ONEDITORLOADED and ONDATALOADED event handler (See "Event Handling with JavaScript"); if not included within the APPLETTAG, the applet will not react in any way to these event handlers.

Important: edit-on Pro 3.x is not 100% compatible with edit-on Pro 2.x because there are a number of differences in the API and in the content format. Documents created with edit-on Pro 2.x or a previous version may not be directly compatible with this version and vice versa. However, edit-on Pro 3.x simplifies the usage of almost all legacy content by providing an integrated conversion mechanism based on W3C's HTML TIDY technology. This process is transparent to the end user. In general, when updating to version 3.x it is also necessary to modify the integration of the applet because of the API changes made in version 3.x.

2 edit-on Pro API Reference

2.1 Applet Parameter API

Following are the descriptions of all applet parameters supported in edit-on Pro:

Function:	LICENSEKEY
Syntax:	<code><PARAM name="LICENSEKEY" value="someotherlicensekey.xml"></code>
Default:	licensekey.xml
Description:	Specifies the name of the license key xml file. The URL may be relative to the applet codebase. If this parameter is not specified, edit-on Pro will look in the applet codebase and load the licensekey.xml file located there. If it fails to find it, then edit-on Pro will not be loaded. See "License Key Mechanism"
Function:	ENCODING
Syntax:	<code><PARAM name="ENCODING" value="UTF8"></code>
Possible Values:	e.g.: UTF8, for possible values see http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html <i>NOTE:</i> Since edit-on Pro is written in Java 1.1, we have always provided the above link in earlier versions of this document, in order to point to the Java 1.1 encoding overview. However, this section of Sun's website has been discontinued. Please be aware that several encodings mentioned on the encoding overview page above may not be supported in Java 1.1. <i>IMPORTANT:</i> This parameter is case-sensitive, this means e.g. "utf8" and "UTF8" are different values.
Description:	Specifies the encoding of the data for export/import. The server side script should handle the incoming data using the encoding specified in this parameter.
Important Note:	The name of a specific encoding may not be the same in the term used as value for the applet's encoding parameter (which is the Java term used for this encoding) and in the HTML term for this encoding, which is set in the page's META tag. The page containing the <applet> tag must use the same encoding as the applet. For example, to use the UTF-8 encoding, "UTF8" has to be set in the applet's encoding parameter, while "utf-8" has to be used in the page's META tag to set its encoding to UTF-8.
	<p>Example:</p> <pre><head> ... <meta http-equiv="Content-Type" content="text/html; charset="utf-8" /> ... </head> ... <applet ... > <PARAM name="ENCODING" value="UTF8"> ... </applet></pre> <p>Additionally, support for specific encodings may be platform dependent.</p>
Function:	EXPORTASNUMERICAL
Syntax:	<code><PARAM name="EXPORTASNUMERICAL" value="true"></code>
Default:	false
Description:	Specifies whether edit-on Pro should export Latin-1 characters, special characters and symbols (as defined in XHTML entity sets) as numeric character entities, e.g. ä will be exported as ä.
Function:	TOOLBAR URL
Syntax:	<code><PARAM name="TOOLBARURL" value="toolbar.xml"></code>

Description: Specifies the name/URL of the toolbar definition file. The name may be relative to the applet codebase. This parameter is required, or else the toolbar will show up empty. This parameter replaces the parameter `BUTTONORDER` (this should no longer be used). For details about the toolbar definition file see "Toolbar Definition".

Function: **CONFIGURL**

Syntax: `<PARAM name="CONFIGURL" value="config.xml">`

Description: Specifies the name/URL of the XML configuration file. The name may be relative to the applet codebase. For details about the file format see "Configuration File".

Function: **TABLENBSPFILL**

Syntax: `<PARAM name="TABLENBSPFILL" value="true">`

Default: false

Description: Table cells which are left empty are filled with the ` ` entity when exported.

Function: **IMAGEROOT**

Syntax: `<PARAM name="IMAGEROOT" value="http://www.realobjects.com/images">`

Default: none

Description: If this parameter is specified, all relative images will then be expanded along with this image root in order to find the image to display. The relative image address will never be changed (by exporting or when the image properties are shown). The name may be relative to the applet document base. If the `IMAGEROOT` parameter is not defined, then image URLs are truncated/expanded during loading and saving of the document. For details see "Image Root".

Function: **UPLOADIMAGE**

Syntax: `<PARAM name="UPLOADIMAGE" value="http://www.realobjects.com/images/upload.jsp">`

Description: Specifies the CGI/ASP/PHP/JSP URL that should receive the image data uploaded by the applet to the server. The request will be a POST request. The image data will be stored in a field with the name "uploadfile". The CGI/ASP/PHP/JSP URL should save the image data into an image file within the server.

Function: **EXPORTROOT URL**

Syntax: `<PARAM name="EXPORTROOTURL" value="http://www.realobjects.com/images">`

Description: If this parameter is specified, all images will be exported according to the URL specified in this parameter. It is only used for direct HTTP connections. This parameter is ignored if the `IMAGEROOT` parameter is defined.

Function: **IMPORTROOT URL**

Syntax: `<PARAM name="IMPORTROOTURL" value="http://www.realobjects.com/images">`

Description: If this parameter is specified, all relative images will be expanded with the URL, in order to find the image to be displayed. The relative image address will never be changed (through exporting or when the image properties are shown). The name may be relative to the applet document base. This parameter works only for import. When this parameter is specified, `IMAGEROOT` is ignored during the data import process.

Function: **SETHTMLDATA URL**

Syntax: `<PARAM name="SETHTMLDATAURL" value="http://www.realobjects.com/test.htm">`

Description: Specifies the URL of the document to be loaded, after the applet has initialized.

Function: **SET_OK_URL**

Syntax: `<PARAM name="SET_OK_URL" value="result.jsp?action=import%20to%20document&result=successful">`

Description: The URL specified in this parameter will be called when the HTML data has

been set successfully into the editor. The URL may be relative to the document base. This parameter will only take effect if the parameter SETHTMLDATAURL is specified.

Function: **SET_OK_TARGET**

Syntax: `<PARAM name="SET_OK_TARGET" value="result">`

Description: The Target URL specified in this parameter will be used as target frame for SET_OK_URL.

Function: **SET_ERROR_URL**

Syntax: `<PARAM name="SET_ERROR_URL" value="result.jsp?action=import%20to%20document&result=failed">`

Description: The URL specified in this parameter will be called when the HTML data has not been set into the editor successfully. The URL might be relative to the document base. This parameter will only take effect if the parameter SETHTMLDATAURL is specified.

Function: **SET_ERROR_TARGET**

Syntax: `<PARAM name="SET_ERROR_TARGET" value="result">`

Description: The Target URL specified in this parameter will be used as target frame for SET_ERROR_URL.

Function: **GETHTMLDATA URL**

Syntax: `<PARAM name="GETHTMLDATAURL" value="http://www.realobjects.com/postdata.asp? ">`

Description: Specifies the CGI/ASP/PHP/JSP URL that should receive the HTML posted by the applet, after clicking on the SAVE button. The request will be a POST request and the data will be stored in a field with the name "HTMLDATA". See also: "Using direct HTTP connections".

Function: **CUSTOMFIELD**

Syntax: `<PARAM name="CUSTOMFIELD" value="CustomFieldValue">`

Description: Used to specify an additional custom form field which will be sent to the CGI/ASP/PHP/JSP URL; the custom field will receive the HTML data posted by the applet, after clicking on the SAVE button. The request will be a POST request and the data will be stored in a field with the name "CUSTOMFIELD". See also: "Using direct HTTP connections".

Function: **GET_OK_URL**

Syntax: `<PARAM name="GET_OK_URL" value="result.jsp?action=send%20to%20server&result=successful">`

Description: The URL specified in this parameter will be called when the HTML data has been successfully sent from the editor, to an URL address specified in the parameter GETHTMLDATAURL. The URL may be relative to the document base. This parameter will only take effect if the parameter GETHTMLDATAURL is specified.

Function: **GET_OK_TARGET**



Syntax: `<PARAM name="GET_OK_TARGET" value="result">`

Description: The Target URL specified in this parameter will be used as target frame for GET_OK_URL.

Function: **GET_ERROR_URL**

Syntax: `<PARAM name="GET_ERROR_URL" value="result.jsp?action=send%20to%20server&result=failed">`

Description: The URL specified in this parameter will be called when the HTML data has not been sent successfully, from the editor to an URL address specified in the parameter GETHTMLDATAURL. The URL may be relative to the document base. This parameter will only take effect if the parameter GETHTMLDATAURL is specified.

- Function: **GET_ERROR_TARGET**
 Syntax: `<PARAM name="GET_ERROR_TARGET" value="result">`
 Description: The Target URL specified in this parameter will be used as target frame for GET_ERROR_URL.
- Function: **LOCALE**
 Syntax: `<PARAM name="LOCALE" value="en_US">`
 Default: en_US
 Possible Value: en_US, de_DE, fr_FR, es_ES
 Description: Specifies the language module, which defines the UI language. For details see "LOCALE Setting".
- Function: **LOCALEURL**
 Syntax: `<PARAM name="LOCALEURL" value="locale_en_US.xml">`
 Description: Specifies the name of the locale file. The value may be relative to the applet codebase. If specified, this parameter overrides the parameter LOCALE. If this parameter is not specified, then edit-on Pro will look in the applet codebase and load the locale file according to the locale parameter. For details see "LOCALE Setting".
- Function: **HELP**
 Syntax: `<PARAM NAME="help" VALUE="eophelp/en_US/help_en_US.htm">`
 Description: Used to specify the URL of the help file in HTML format. The help will be displayed in a new browser window, when the  icon is clicked. The  icon must be activated by setting the HELP parameter in the toolbar definition file. See "Toolbar Definition".
- Function: **SOURCEVIEW**
 Syntax: `<PARAM name="SourceView" value="true">`
 Default: false
 Description: When true, it is possible to switch from the "WYSIWYG view" to the "Source View" where the HTML code can be edited. In the "Source View", most toolbar buttons will be disabled. When switching back to the "WYSIWYG View", the HTML is parsed again and displayed accordingly. When false, the "Source View" is not available.
- Function: **SOURCEVIEWWORDWRAP**
 Syntax: `<PARAM name="SOURCEVIEWWORDWRAP" value="true">`
 Default: false
 Description: If the value is true, word wrapping is enabled in the source view. This parameter will only take effect if the SOURCEVIEW parameter value is true.
- Function: **BODYONLY**
 Syntax: `<PARAM name="BODYONLY" value="true">`
 Default: false
 Description: When true, the editor will not process (on load and save, or get and set) header, title, body etc. tags. It will only concern itself with the HTML "body". When false, the editor will read and preserve header tags, title tags etc. It will deliver a complete HTML page when exporting. Always use BODYONLY=true when dealing with fragments of HTML pages.
- Function: **NBSPFILL**
 Syntax: `<PARAM name="NBSPFILL" value="true">`
 Default: false
 Description: On export empty paragraphs, outside tables are filled with the entity. To make sure all paragraphs are filled with the entity, make sure that the TABLENBSPFILL and NBSPFILL parameters are set to true.

- Function:** (UI Colors)
- Syntax:**

```
<PARAM name="WINDOWFACECOLOR" value="#000099">
<PARAM name="TABPANEACTIVECOLOR" value="#EEEEEE">
<PARAM name="WINDOWHIGHLIGHTCOLOR" value="#FFCC66">
<PARAM name="LIGHTEDEGECOLOR" value="#FF0000">
<PARAM name="DARKEDGECECOLOR" value="#FF0000">
<PARAM name="INNERTEXTCOLOR" value="#0000CC">
<PARAM name="STARTUPSCREENBACKGROUNDCECOLOR" value="#FFCC66">
<PARAM name="STARTUPSCREENTEXTCECOLOR" value="#0000A0">
<PARAM name="DISABLEDICONCECOLOR" value="#FFFFFF">
```
- Description:** Customizing of the applet UI colors. Use Hex RGB values to set the colors
- Function:** **STYLESHEETURL**
- Syntax:**

```
<PARAM name="STYLESHEETURL"
value="http://www.realobjects.com/styles.css">
```
- Description:** Specifies the URL of the style sheet file, to be loaded after the applet has initialized.
- Function:** **CUSTOMCOLORSONLY**
- Syntax:**

```
<PARAM name="CUSTOMCOLORSONLY" value="true">
```
- Default:** false
- Description:** If the value is true, the color dialogs used in edit-on Pro will only display custom colors. Standard colors will not be displayed.
- Function:** (Custom Dialogs)
- Syntax:**

```
<PARAM name="insertimagedlg" value="com.my.dialog">
<PARAM name="editimagedlg" value="com.my.dialog">
<PARAM name="inserthrefdlg" value="com.my.dialog">
<PARAM name="edithrefdlg" value="com.my.dialog">
<PARAM name="openhtmldlg" value="com.my.dialog">
<PARAM name="inserthtmldlg" value="com.my.dialog">
<PARAM name="editunknowntagdlg" value="com.my.dialog">
<PARAM name="tablepropertiesdlg" value="com.my.dialog">
<PARAM name="rowpropertiesdlg" value="com.my.dialog">
<PARAM name="columnpropertiesdlg" value="com.my.dialog">
<PARAM name="insettabledlg" value="com.my.dialog">
<PARAM name="colordlg" value="com.my.dialog">
<PARAM name="insertspandlg" value="com.my.dialog">
```
- Default:** No Custom Dialogs
- Description:** When defined, the specified class will be called to invoke the custom dialog in place of the standard implementation. For details see "Custom Dialog API".
- Function:** **NEWDOCUMENTDIALOG**
- Syntax:**

```
<PARAM name="NEWDOCUMENTDIALOG" value="true">
```
- Default:** false
- Description:** When true, the editor will display a dialog box when the "new document" button is clicked. The dialog asks for confirmation before clearing the editor content and creating a new document. When false, the editor will not display the dialog box, and will clear the editor content without asking for confirmation.
- Function:** **COMBOFONTSIZE**
- Syntax:**

```
<PARAM name="COMBOFONTSIZE" value="12">
```
- Description:** Specifies the font size in the combo boxes of the editor's toolbar. When this parameter is not specified, the size of the fonts in the combo boxes of the editor's toolbar, will not be set by edit-on Pro.
- Function:** **ONEDITORLOADED**
- Syntax:**

```
<PARAM name="ONEDITORLOADED" value="OnEditorLoaded">
```
- Description:** Specifies the JavaScript function that will be executed when the editor finishes its loading event. See "Event Handling with JavaScript". Requires the MAYSCRIPT attribute in the <applet> tag to be set.
- Function:** **ONDATALOADED**
- Syntax:**

```
<PARAM name="ONDATALOADED" value="OnDataLoaded">
```

Description: Specifies the JavaScript function that will be executed when the setHTMLData() JavaScript function finishes its data loading event. See “JavaScript API and Event Handling with JavaScript”. Requires the MAYSCRIPT attribute in the <applet> tag to be set.

Function: **ONDATAPOSTED**

Syntax: <PARAM name="ONDATAPOSTED" value="OnDataPosted">

Description: Specifies the JavaScript function that will be executed when HTML data has been successfully sent from the editor to an URL address specified in the parameter GETHTMLDATAURL.

Function: **ONDATAPOSTEDERROR**

Syntax: <PARAM name="ONDATAPOSTEDERROR" value="OnDataPostedError">

Description: Specifies the JavaScript function that will be executed when HTML data has not been successfully sent from the editor to an URL address specified in the parameter GETHTMLDATAURL.

Function: **MULTIPLEUNDOREDO**

Syntax: <PARAM name="MULTIPLEUNDOREDO" value="true">

Default: false

Description: Specifies whether the “multiple undo redo” feature is active or inactive. This enables the user to undo/redo several last actions at once.

Function: **MAXUNDOREDOSTEPS**

Syntax: <PARAM name="MAXUNDOREDOSTEPS" value="20">

Default: 10

Description: Specifies the maximum number of undo/redo steps available in edit-on Pro. Memory usage should be taken into consideration when setting this parameter. The larger the number of available undo/redo steps, the more memory will be used for the undo/redo buffer.

Function: **OLDFONTSTYLEMODE**

Syntax: <PARAM name="OLDFONTSTYLEMODE" value="true">

Default: false

Description: Specifies whether edit-on Pro uses the “old font style” mode or not. When the old font style mode is used (this parameter is set to true), edit-on Pro will use the , <i> and <u> tags for bold, italic and underline tags respectively. When this parameter is set to false, , , and will be used in place of those tags respectively.

Function: **CLASSCOMBOWIDTH**

Syntax: <PARAM name="CLASSCOMBOWIDTH" value="80">

Description: Specifies the width of the style sheet class combo box in edit-on Pro toolbar.

Function: **STANDALONEMODE**

Syntax: <param name="STANDALONEMODE" value="true" />

Default: false

Description: When this parameter is true, the editor is displayed in frame window mode per default.

Function: **SMARTINDENT**

Syntax: <PARAM name="SMARTINDENT" value="false">

Default: true

Description: Specifies whether the smart indent feature is active or not. Smart indent enables the editor to export HTML data in a clean indented layout, thus making the exported HTML document easier to understand.

Function: **LCE**

Syntax: <PARAM name="LCE" value="MACIE">

Description: Specifies whether the LiveConnect Emulation (LCE) is intended to be used on a

Mac client together with the Microsoft Internet Explorer for Mac as browser. The only possible value is "MACIE". Using LiveConnect Emulation means that the JavaScript API for Mac is used instead of the JavaScript API that already has been provided with older versions of edit-on Pro. When setting this parameter's value to "MACIE", you must use the JavaScript API for Mac. For details on how to integrate this API see section "JavaScript API for Mac OS".

Note: The Applet will not load if the parameter "TOOLBARURL" is not set, or if the toolbar loaded via this parameter contains errors. In both cases, the applet will display an error message and will stop loading. For additional information, please review the chapter Toolbar Definition.

Additionally, it is highly recommend to set the parameter "CONFIGURL". If this parameter is not set, the applet will load a default configuration. If it is set, the applet will also display an error message and stop loading if there are errors in the configuration file. For additional information, please review the chapter Configuration File.

2.2 Configuration File

2.2.1 XML Format

The configuration file specified by the parameter CONFIGURL uses the XML format to specify the configuration settings of edit-on Pro.

The XML configuration file specifies these settings for edit-on Pro:

1. fontmapping
2. presetcolors
3. customtags
4. spellingcheckers
5. cleanupprocess
6. defaulttablesettings
7. inserttext_html
8. htmlformatclipboard
9. paragraphstyle
10. toolbar
11. alternateenterkeyaction
12. fontsizeunit
13. fontsizes

- **fontmapping**
Specifies to which fonts the rendered fonts will be mapped when they are exported. The mapped fonts are specified using `font` tags:

`font`

Specifies each mapping of fonts within the editor.

Required information:

- Name: displayed name within the editor, this name will be displayed in toolbar's font choice

- Rendered font: specifies the font name used to render this font mapping
- Exported font: specifies the font names used to export this font mapping

Example:

```
<fontmapping>
  <font name="Roman">
    <renderedfont>Serif</renderedfont>
    <exportedfont>Georgia, Times New Roman, Times, Serif</exportedfont>
  </font>
</fontmapping>
```

- **presetcolors**
Specifies the color presets used in the custom color section of the color. The color presets are specified using `color` tags.

`color`
Specifies each color preset.
Required information:
- `rgb`: RGB value of the color
Optional Information:
- `desc`: description of the color

Example:

```
<presetcolors>
  <color rgb="#FF0000" desc="red" />
</presetcolors>
```

- **customtags**
Specifies which custom tags can be used within the editor. The custom tags are specified using either `customtag` or `emptycustomtag`. `emptycustomtag` is a special custom tag that does not have any content:

`customtag`
Specifies each non-empty custom tag.
Required information:
- `name` : the name of the custom tag
- `icons`: specifies the icons used to display this custom tag. Use `starticon` to define the icon used for the opening tag and `endicon` to define the icon used to display the closing tag:
 - `starticon`
 - `endicon`
Optional Information:
- `alwaysshow` (Default: false)
- `isreadonlytag` (Default: false)
- `isreadonlyarea` (Default: false)
- `isblockelement` (Default: false)
- `dlgclassname`: to specify which class is used when this tag's properties event is invoked.

`emptycustomtag`
Specifies each empty custom tag.
Required information :
- `name` : the name of the custom tag
- `icon`: specifies the icon use to display this custom tag
Optional Information :
- `alwaysshow` (Default : false)
- `isreadonlytag` (Default : false)
- `isreadonlyarea` (Default : false)
- `isblockelement` (Default : false)
- `dlgclassname`: to specify which class is used when this tag's properties event is invoked.

Example :

```
<customtags>
  <customtag name="Headline">
    <icons>
```

```

        <starticon>sampleicons/headline.gif</starticon>
        <endicon>sampleicons/headline.gif</endicon>
    </icons>
    <dlgclassname>sampledlg.headlineTagProperties</dlgclassname>
</customtag>
<emptycustomtag name="Language">
    <icon>sampleicons/language.gif</icon>
    <dlgclassname>sampledlg.headlineTagProperties</dlgclassname>
</emptycustomtag>
</customtags>

```

- **spellingcheckers**

Specifies the languages available in the spelling checker. The spelling checker languages are specified using `spellingchecker` tags. To enable the user to add new words to the spelling checker dictionary, please use `addspellcheckwordurl`

`spellingchecker`

Specifies each language available.

Required information:

- name : language name
- properties : language properties file

Optional Information:

- default (Default : false)

`addspellcheckwordurl`

Specifies the CGI/ASP/PHP/JSP URL that should receive the word data that will be posted by the applet after clicking on the “add word” button in the spell check dialog. The request will be a POST request. The word will be contained in a field with the name “WORD”, and the currently active spell check language will be contained in a field with the name “LANG”. The CGI/ASP/PHP/JSP URL should append the new word to the related text lexicon file. The new word will be recognized next time the spell check action is performed. When this tag is not specified, or is specified with an empty url attribute, the “add word” button in the spell check dialog will be disabled. See “Creating a Custom Lexicon (Dictionary)”.

Optional Information:

- url: add spell check word url (Default : empty)

Example :

```

<spellingcheckers>
    <spellingchecker name="english" properties="english.properties" default="true"/>
    <addspellcheckwordurl url="http://www.realobjects.com/addword.jsp"/>
</spellingcheckers>

```

- **cleanupprocess**

Specifies the url of the server- side clean- up process script, and/or client-side process class that is required to assure the wellformedness of the document loaded into the editor

Required information :

- url : server side clean up process url
- class : client side clean up process class

Optional Information:

- isclientside (Default : false)
- enabled (Default : false)
- alwaysclean (Default : false)
- tidyconfigfile

Example :

```

<cleanupprocess url="http://MyWebServer/CleanUpProcess.jsp" isclientside="false" -
tidyconfigfile="http://localhost/eop/tidyconfig.txt" enabled="true"/>

```

- **defaultablesettings**

Specifies the default settings of the standard table, which will be inserted when the button “insert table” is clicked.

Optional Information :

- Table_NumRows: specifies the number of rows of the standard table. Default value is 2
- Table_NumCols: specifies the number of columns of the standard table. Default value is 2

- Table_OverallWidth: specifies the width of the standard table. Default value is 100%
- Table_RowHeight: specifies the height of the rows of the standard table. Default value is 20
- Table_ColWidth: specifies the width of columns of the standard table. Default value is 20%

Example :

```
<defaulttablesettings>
<table_numrows>3</table_numrows>
<table_numcols>4</table_numcols>
<table_overallwidth>100%</table_overallwidth>
<table_rowheight>10</table_rowheight>
<table_colwidth>25%</table_colwidth>
</defaulttablesettings>
```

- inserttext_html
Specifies whether the HTML option in the insert text dialog is activated or not. This setting will only affect if the INSERTTEXT button is activated in the toolbar definition file.

Optional Information:

- enabled (Default : false)

Example :

```
<inserttext_html enabled="true"/>
```

- htmlformatclipboard
Specifies whether or not the capability to paste content from or copy content to an application that is able to export html to the clipboard is activated. Please note that this setting will only work when using the JRE 1.4.1 on the Windows platform. Applications that are able to export their content as html to the clipboard are e.g. Microsoft Word, Microsoft Excel, Microsoft Internet Explorer or OpenOffice. There may be others, however, but the ones mentioned here have been tested.

Optional Information:

- enabled (Default : false)

Example :

```
<htmlformatclipboard enabled="true"/>
```

IMPORTANT: To make the htmlformatclipboard option work, "tidy.jar" and "tidy.cab" must be loaded in the applet. If they are not loaded, the clean up can not be performed. In order to achieve this, tidy archives ("tidy.jar" and "tidy.cab") have to be added to the *cabbase* and *archive* applet parameters.

- paragraphstyles
Specifies the styles that can be set in the paragraph style choice box. The paragraph styles are specified using `paragraphstyle` tags:

`paragraphstyle`

Specifies the individual paragraph styles available in the paragraph style choice box.

Required information:

- name: specifies the name of the individual paragraph styles.

Example :

```
<paragraphstyles>
  <paragraphstyle name="Standard" />
  <paragraphstyle name="H1" />
  <paragraphstyle name="H2" />
  <paragraphstyle name="H3" />
  ...
</paragraphstyles>
```

Note: The paragraph style "Standard" is a default setting that is always set within the paragraph style choice box whether it is defined within the config.xml or not.

- alternateenterkeyaction
Specifies which line break tags should be generated when the enter key is pressed.
- Optional Information:

- enabled (Default : false): when this option is true, the applet will generate `
` tags instead of `<p>` tags when the enter key is pressed, and generate `<p>` tags instead of `
` tags when shift+enter is pressed.

Example :

```
<alternateenterkeyaction enabled="true" />
```

- `fontsizeunit`
Specifies which unit should be used to display fonts per default.
- unit (Default : "px"): the possible values are "pt" and "px"

Example :

```
<fontsizeunit unit="pt" />
```

- `fontsizes`
Specifies the font sizes that are available in the font size choice box. The font sizes are specified using `fontsize` tags.

`fontsize`

Specifies the individual font sizes available in the font size choice box.

Required information:

- size: specifies the size of the individual font size.

Example:

```
<fontsizes>
  <fontsize size="12" />
  <fontsize size="14" />
  <fontsize size="36" />
</fontsizes>
```

2.2.2 Sample Configuration File in XML Format

```
<config>
  <fontmapping>
    <font name="SubHeadline">
      <renderedfont>Serif</renderedfont>
      <exportedfont>georgia, times new roman, times, serif</exportedfont>
    </font>
    <font name="Headline">
      <renderedfont>Sans Serif</renderedfont>
      <exportedfont>arial, helvetica, verdana, sansserif</exportedfont>
    </font>
    <font name="Description">
      <renderedfont>Monospaced</renderedfont>
      <exportedfont>courier, courier new, monospaced</exportedfont>
    </font>
    <font name="News">
      <renderedfont>Sans Serif</renderedfont>
      <exportedfont>arial, helvetica, verdana, sansserif</exportedfont>
    </font>
    <font name="Reporter">
      <renderedfont>Serif</renderedfont>
      <exportedfont>georgia, times new roman, times, serif</exportedfont>
    </font>
  </fontmapping>
  <presetcolors>
    <color rgb="#0000FF" desc="Subheadline" />
    <color rgb="#FF0000" desc="Headline" />
    <color rgb="#2789ff" desc="Description" />
    <color rgb="#808080" desc="News" />
    <color rgb="#008000" desc="Reporter" />
  </presetcolors>
  <customtags>
    <customtag name="headline" alwaysshow="true" isreadonlytag="true"
      isblockelement="true">
      <icons>
        <starticon></starticon>
        <endicon></endicon>
      </icons>
      <dlgclassname>sampldlg.warningTagProperties</dlgclassname>
    </customtag>
```

```

    <customtag name="description" alwaysshow="true">
      <icons>
        <starticon>sampleicon/description.gif</starticon>
        <endicon>sampleicon/description.gif</endicon>
      </icons>
      <dlgclassname>sampledlg.descriptionTagProperties</dlgclassname>
    </customtag>
    <customtag name="news" alwaysshow="true">
      <icons>
        <starticon>sampleicon/news.gif</starticon>
        <endicon>sampleicon/news-close.gif</endicon>
      </icons>
      <dlgclassname>sampledlg.warningTagProperties</dlgclassname>
    </customtag>
    <customtag name="reporter" alwaysshow="true">
      <icons>
        <starticon>sampleicon/reporter.gif</starticon>
        <endicon>sampleicon/reporter-close.gif</endicon>
      </icons>
      <dlgclassname>sampledlg.reporterTagProperties</dlgclassname>
    </customtag>
    <emptycustomtag name="topic" alwaysshow="false" isreadonlytag="true">
      <icon>sampleicon/topic.gif</icon>
      <dlgclassname>sampledlg.topicTagProperties</dlgclassname>
    </emptycustomtag>
    <emptycustomtag name="language" alwaysshow="false">
      <icon>sampleicon/language.gif</icon>
      <dlgclassname>sampledlg.languageTagProperties</dlgclassname>
    </emptycustomtag>
    <emptycustomtag name="date" alwaysshow="false">
      <icon>sampleicon/date.gif</icon>
      <dlgclassname>sampledlg.warningTagProperties</dlgclassname>
    </emptycustomtag>
  </customtags>
  <spellingcheckers>
    <spellingchecker name="English" properties="english.properties"
      default="true"/>
    <spellingchecker name="German" properties="german.properties"
      default="false"/>
  </spellingcheckers>
  <cleanupprocess isclientside="true" enabled="true"
    tidyconfigfile="http://localhost/eop/tidyconfig.txt" alwaysclean="true"/>
  <defaulttablesettings>
    <table_numrows>3</table_numrows>
    <table_numcols>4</table_numcols>
    <table_overallwidth>100%</table_overallwidth>
    <table_rowheight>10</table_rowheight>
    <table_colwidth>25%</table_colwidth>
  </defaulttablesettings>
  <inserttext_html enabled="true"/>
  <htmlformatclipboard enabled="true"/>
  <paragraphstyles>
    <paragraphstyle name="Standard" />
    <paragraphstyle name="H1" />
    <paragraphstyle name="H2" />
    <paragraphstyle name="H3" />
    <paragraphstyle name="H4" />
    <paragraphstyle name="H5" />
    <paragraphstyle name="H6" />
    <paragraphstyle name="PRE" />
    <paragraphstyle name="DIV" />
  </paragraphstyles>
  <alternateenterkeyaction enabled="true" />
  <fontsizeunit unit="pt" />
  <fontsizes>
    <fontsize size="12" />
    <fontsize size="14" />
    <fontsize size="36" />
  </fontsizes>
</config>

```

Note: If the `cleanupprocess` parameter is enabled "tidy.jar" and "tidy.cab" must be loaded in the applet. If they are not loaded, the clean up can not be performed. In order to achieve this, tidy archives ("tidy.jar" and "tidy.cab") have to be added to the `cabbase` and `archive` applet parameters.

2.3 Toolbar Definition

2.3.1 XML Format

The toolbar definition, specified by the parameter TOOLBARURL, uses the XML format to enable/disable the functions available in the toolbar, and the order of the buttons and the icons used.

The XML format for toolbar definition supports six tag types within the *Toolbar* tag as parent element, they are:

1. icon
 2. subicon
 3. choice
 4. separator
 5. endrow
 6. custom button (see “Custom Button”)
- **Toolbar**
Specifies if the toolbar should be displayed. The toolbar can either be collapsed or extended by default.
Optional Information:
- default (Default : false)

Example : `<toolbar collapse="true" />`

- **Icon**
Required information:
- Parameter name : button identity
Optional Information:
- Enabled state (Default : true)
- Default list (Default : None)
- Icon file (each parameter has a default icon file)

Allowed Parameter Names:

NEW, LOAD, GETHTMLDATAURL, CUT, COPY, PASTE, BOLD, ITALIC, UNDERLINE, COLOR, ALIGN, ORDEREDLIST, UNORDEREDLIST, INCIDENT, DECIDENT, LINK, BOOKMARK, IMAGE, INSERTHTML, INSERTTEXT, HR, SPECIALCHARACTER, SHOWNALLCHAR, PROPERTIESDIALOG, INSERTTABLE, INSERTTABLEDIALOG, INSERTROW, INSERTCOLUMN, SPELLCHECK, HELP, INFO

Allowed Default Lists (The Default Lists will only be processed if the parameter name of the icon is ORDEREDLIST or UNORDEREDLIST):

DISCUNORDEREDLIST, CIRCLEUNORDEREDLIST, SQUAREUNORDEREDLIST, LOWERALPHAORDEREDLIST, UPPERALPHAORDEREDLIST, NUMBEREDORDEREDLIST

Example :

```
<icon paramname="INSERTHTML" enabled="false">
  <iconfile>icons/file.gif</iconfile>
</icon>
```

- **Subicon**
Required information:
- Parameter name : button identity
Optional Information:

- Enabled state (Default : true)
- Icon file (each parameter has a default icon file)

Allowed Parameter Names:

DISCUNORDEREDLIST, CIRCLEUNORDEREDLIST, SQUAREUNORDEREDLIST,
LOWERALPHAORDEREDLIST, UPPERALPHAORDEREDLIST, NUMBEREDORDEREDLIST

Example :

```
<icon paramname="UNORDEREDLIST">
  <iconfile>icons/disculist.gif</iconfile>
  <subicon paramname="discunorderedList">
    <iconfile>icons/disculist.gif</iconfile>
  </subicon>
  <subicon paramname="circleunorderedList">
    <iconfile>icons/circleulist.gif</iconfile>
  </subicon>
  <subicon paramname="squareunorderedList">
    <iconfile>icons/squareulist.gif</iconfile>
  </subicon>
</icon>
```

- Choice

Required information:

- Parameter name : button identity

Optional Information:

- Enabled state (Default : true)

Allowed Parameter Name:

PARAGRAPHSTYLE, FONT, FONTSIZE

Example:

```
<choice paramname="PARAGRAPHSTYLE" enabled="true"/>
```


- Separator


Use the `<separator />` tag to separate each button group.

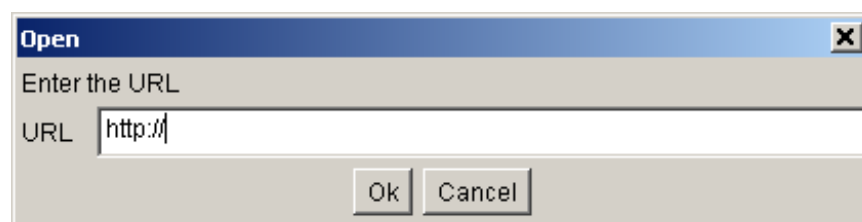
- Endrow

Use the `<endrow />` tag to force a new row in the toolbar.


2.3.2 Functions


Parameter: **NEW**
 Syntax: NEW;Icon_URL
 Default Icon: 
 Default Icon URL: icons/new.gif
 Description: When clicked, all content in the current document will be deleted.


Parameter: **LOAD**
 Syntax: LOAD;Icon_URL
 Default Icon: 
 Default Icon URL: icons/open.gif
 Description: When clicked, a dialog box will ask for a URL. After pressing OK, the editor will retrieve the respective HTML file or Web page, and then display it.




Dialog Screenshot:

Parameter: **GETHTMLDATAURL**
 Syntax: GETHTMLDATAURL;Icon_URL
 Default Icon: 
 Default Icon URL: icons/save.gif
 Description: When clicked, the editor will post the HTML data to the URL specified by the applet parameter GETHTMLDATAURL.
 Required: Parameter GETHTMLDATAURL within the applet.

Parameter: **CUT, COPY, PASTE**
 Syntax: CUT; Icon_URL
 COPY;Icon_URL
 PASTE;Icon_URL
 Default Icon: 
 Default Icon URL: icons/cut.gif, icons/copy.gif, icons/paste.gif
 Description: The well known cut /copy & paste functionality.

Parameter: **UNDO, REDO**
 Syntax: UNDO; Icon_URL
 REDO; Icon_URL
 Default Icon: 
 Default Icon URL: icons/undo.gif, icons/redo.gif
 Description: Allows the user to undo/redo several last actions. When the MUTIPLEUNDOREDO applet parameter is set to true, the user can also undo/redo several last actions in one click. The number of undoable/redoable actions is specified by using the MAXUNDOREDOSTEPS applet parameter (default steps is 10)

Parameter: **FIND**
 Syntax: FIND
 Default Icon: 
 Default Icon URL: icons/find.gif
 Description: Replaces one or all occurrences of the specified text in the document. The available options include Match Case and Direction options.



Dialog Screenshot:

Parameter: **PARAGRAPHSTYLE**
 Syntax: PARAGRAPHSTYLE
 Description: Different paragraph styles can be selected in the drop down list.

Parameter: **FONT**
 Syntax: FONT
 Description: The available fonts can be configured in the congiguration file. See "Configuration File".

Parameter: **FONTSIZE**
 Syntax: FONTSIZE
 Description: Different font sizes can be selected in the drop down list.

Parameter: **STYLESHEET**
 Syntax: STYLESHEET
 Description: Different stylesheet classes can be selected in the drop down list.


Parameter: **BOLD, ITALIC, UNDERLINE**
 Syntax: BOLD;Icon_URL
 ITALIC;Icon_URL
 UNDERLINE;Icon_URL
 Default Icon: **B I U**
 Default Icon URLs: icons/bold.gif, icons/italic.gif, icons/underline.gif
 Description: Common font styles which can be combined.


Parameter: **SUBSCRIPT, SUPERScript**
 Syntax: SUBSCRIPT;Icon_URL
 SUPERScript;Icon_URL
 Default Icon: **x₂ x²**
 Default Icon URLs: icons/subscript.gif, icons/superscript.gif
 Description: Allows typing of subscript/superscript text.

Parameter: **COLOR**
 Syntax: COLOR;Icon_URL
 Default Icon: 
 Default Icon URL: icons/autocolor.gif
 Description: Allows the user to select different text colors by choosing a color from the color selection dialog box. In addition to the standard color palette, the user can define custom colors as well.




Dialog Screenshot:

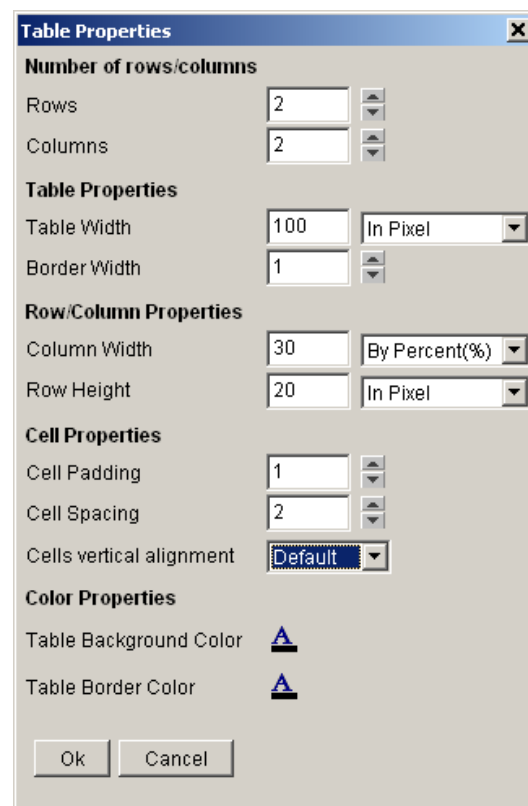
Parameter: **ALIGNLEFT, ALIGNCENTER, ALIGNRIGHT, ALIGNJUSTIFY**
 Syntax: ALIGNLEFT;left_icon_URL
 ALIGNCENTER:center_icon_URL
 ALIGNRIGHT:right_icon_URL
 ALIGNJUSTIFY;justify_icon_URL
 Default Icons: 
 Default Icon URLs: icons/left.gif, icons/center.gif, icons/right.gif, icons/justify.gif
 Description: Allows the user to change the alignment of paragraphs. This will always apply to the whole paragraph Each icon can be individually enabled or disabled. Please note that the justified alignment is rendered in the same way as the left alignment, but exported to XHTML code as align="justify".

Parameter: **ALIGN (Deprecated)**
 Parameter: ALIGN;left_icon_URL:center_icon_URL:right_icon_URL;justify_icon_URL
 Default Icon: 
 Default Icon URLs: icons/left.gif, icons/center.gif, icons/right.gif, icons/justify.gif
 Description: Provides the option to change the alignment of paragraphs. This will always apply to the whole paragraph. Please also note that the justified alignment is rendered in the same way as the left alignment, but exported to XHTML code as


align="justify".

Note: The ALIGN parameter is only part of the applet to ensure backwards compatibility with older versions of edit-on Pro 3.x. It should not be used anymore. Please do not use the ALIGN parameter in conjunction with any of the individual alignment icons (ALIGNLEFT, ALIGNCENTER, ALIGNRIGHT, ALIGNJUSTIFY).

Parameter:	INSERTTABLEDIALOG
Syntax:	INSERTTABLEDIALOG;Icon_URL
Default Icon:	
Default Icon URL:	icons/table2.gif
Description:	Displays a dialog for setting table properties and inserting a table at the current cursor position.
Required:	The value of the applet parameter SIMPLETABLE must be explicitly set to "TRUE", otherwise this parameter is ignored.
Restriction:	The number of rows and columns that can be inserted is restricted to 10

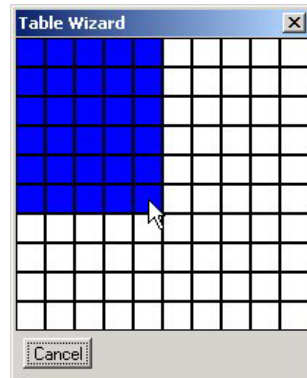


Dialog Screenshot:

Parameter:	INSERTTABLE
Syntax:	INSERTTABLE;Icon_URL
Default Icon:	
Default Icon URL:	icons/table.gif
Description:	Inserts a table at the current cursor position. The inserted table uses the values of the applet parameters TABLENBSPFILL, TABLE_NUMROWS, TABLE_NUMCOLS, TABLE_OVERALLWIDTH, TABLE_ROWHEIGHT and TABLE_COLWIDTH to specify its behaviour.
Required:	The value of the applet parameter SIMPLETABLE must be explicitly set to "TRUE", otherwise this parameter is ignored.

Parameter:	INSERTTABLEWIZARD
Syntax:	INSERTTABLEWIZARD;Icon_URL
Default Icon:	
Default Icon URL:	icons/tablewizard.gif

- Description:** Inserts a table at the current cursor position using the number of rows and columns specified by the mouse position in the dialog. The table inserted uses the applet parameters for `TABLENBSPFILL`, `TABLE_OVERALLWIDTH`, `TABLE_ROWHEIGHT` and `TABLE_COLWIDTH` to specify its behavior.
- Required:** The value of the applet parameter `SIMPLETABLE` must be explicitly set to "TRUE", otherwise this parameter is ignored.
- Restriction:** The number of rows and columns that can be inserted is restricted to 10



Dialog Screenshot:









- Parameter:** **INSERTROW**
- Syntax:** INSERTROW;Icon_URL
- Default Icon:**
- Default Icon URL:** icons/insertrow.gif
- Description:** Inserts a row into a table at the current cursor position.
- Required:** The value of the applet parameter `SIMPLETABLE` must be explicitly set to "TRUE", otherwise this parameter is ignored.



- Parameter:** **INSERTCOLUMN**
- Syntax:** INSERTCOLUMN;Icon_URL
- Default Icon:**
- Default Icon URL:** icons/insertcolumn.gif
- Description:** Inserts a column into a table at the current cursor position.
- Required:** The value of the applet parameter `SIMPLETABLE` must be explicitly set to "TRUE", otherwise this parameter is ignored.


- Parameter:** **INSERTCUSTOMTAG**
- Syntax:** INSERTCUSTOMTAG;Icon_URL
- Default Icon:**
- Description:** Used to insert custom tags defined in the XML configuration file. See "Configuration File".



Dialog Screenshot:

Parameter:	UNORDEREDLIST
Syntax:	UNORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/disculist.gif
Description:	Displays a bullet in front of each list item. edit-on Pro supports three unordered list types: disc, square and circle. Use the button on the right side of the ordered list button to select the list type. Alternatively the style that defines the list type can be specified in the “Style Properties” dialog. See “Style Properties Dialog” and “Styles supported by edit-on Pro” for details.
Parameter:	DISCUNORDEREDLIST
Syntax:	DISCUNORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/disculist.gif
Description:	Displays a disc list in front of each list item.
Parameter:	CIRCLEUNORDEREDLIST
Syntax:	CIRCLEUNORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/circleulist.gif
Description:	Displays a circle list in front of each list item.
Parameter:	SQUAREUNORDEREDLIST
Syntax:	SQUAREUNORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/squareulist.gif
Description:	Displays a square list in front of each list item.
Parameter:	ORDEREDLIST
Syntax:	ORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/numberolist.gif
Description:	Displays an ordered list in front of each list item. edit-on Pro supports three ordered list types: namely Arabic numeric type (“1”), lowercase type (“a”) and uppercase type (“A”). Use the button on the right side of the ordered list button to select the list type. Alternatively the style that defines the list type can be specified in the “Style Properties” dialog. See “Style Properties Dialog” and “Styles supported by edit-on Pro” for details.
Parameter:	LOWERALPHAORDEREDLIST
Syntax:	LOWERALPHAORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/lowerolist.gif
Description:	Displays an ordered list of lowercase type (“a”) in front of each list item.
Parameter:	UPPERALPHAORDEREDLIST
Syntax:	UPPERALPHAORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/upperolist.gif
Description:	Displays an ordered list of uppercase type (“A”) in front of each list item.
Parameter:	NUMBEREDORDEREDLIST
Syntax:	NUMBEREDORDEREDLIST;Icon_URL
Default Icon:	
Default Icon URL:	icons/numberolist.gif
Description:	Displays an ordered list of Arabic numbers (“1”) in front of each list item.

Parameter: **DECINDENT, INCINDENT**
 Syntax: DECINDENT;Icon_URL
 INCINDENT;Icon_URL
 Default Icon:  
 Default Icon URL: icons/decindent.gif, icons/incindent.gif,
 Description: Decreases or increases the level of indentation of a paragraph. Works best together with the list styles. These buttons can also be used to create nested lists.

Parameter: **LINK**
 Syntax: LINK;Icon_URL
 Default Icon: 
 Default Icon URL: icons/link.gif
 Description: Inserts a hyperlink at the current cursor location or converts the currently selected text into a hyperlink. A dialog box will prompt the user to enter the hypertext reference (Href) and the attribute specifications (Target). The equivalent HTML command is `...`.

The TARGET attribute is used with frames, to specify in which frame the link should be opened. Usually, frame names begin with an underscore:

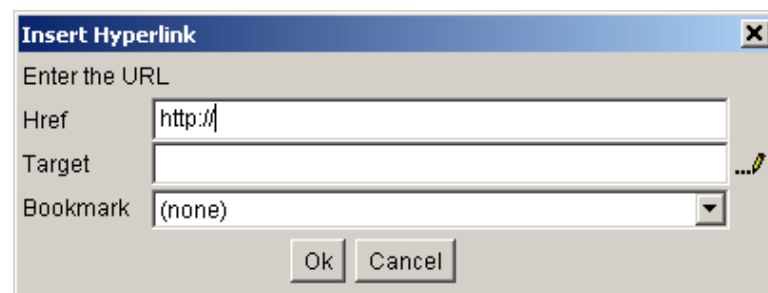
_blank renders the link in a new, unnamed window

_self renders the link in the current frame


_parent renders the link in the immediate FRAMESET parent

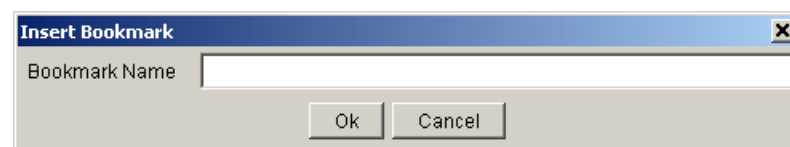
_top renders the link in the full, unframed window

The bookmark choice menu provides the interface to set the hyperlink to a bookmark in the current page. Please note that when a bookmark is selected, the Href textfield will display the corresponding bookmark and when anything is typed in the "Href" textfield, the bookmark choice menu will be reset to (none).




Dialog Screenshot:

Parameter: **BOOKMARK**
 Syntax: BOOKMARK;Icon_URL
 Default Icon: 
 Default Icon URL: icons/bookmark.gif
 Description: Inserts a bookmark at the current cursor location or turns the currently selected text into a bookmark. A dialog box will prompt the user to enter the bookmark name.

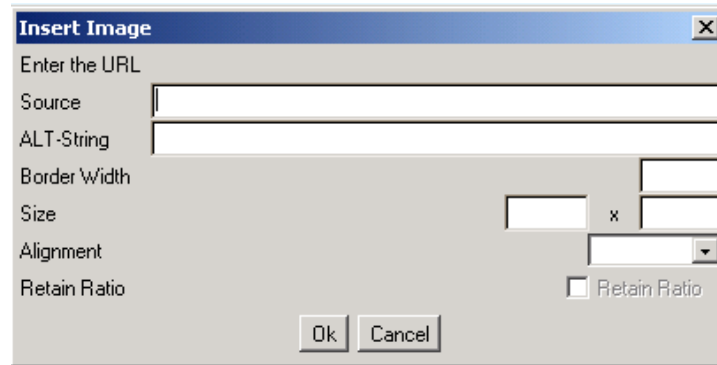


Dialog Screenshot:


Parameter: **IMAGE**
 Syntax: IMAGE;Icon_URL
 Default Icon: 
 Default Icon URL: icons/image.gif
 Description: Inserts an image at the current cursor position. A dialog box will prompt the user

for the image URL, and the attribute specifications for alternate text (ALT), image width (WIDTH), image height (HEIGHT), the width of the image's border (BORDER), the alignment of the image related to text and whether the ratio should be retained. The image will then be displayed at the current cursor location.

Note: The alignment choice can only be used to specify the alignment attribute of the image. edit-on Pro will take this attribute into account when rendering the image.




Dialog Screenshot:

Parameter: INSERTHTML
Syntax: INSERTHTML;Icon_URL
Default Icon: 
Default Icon URL: icons/file.gif
Description: When clicked, a dialog box will ask for an URL. After pressing OK, the editor will retrieve the HTML file or Web page indicated, and insert it at the current cursor position.




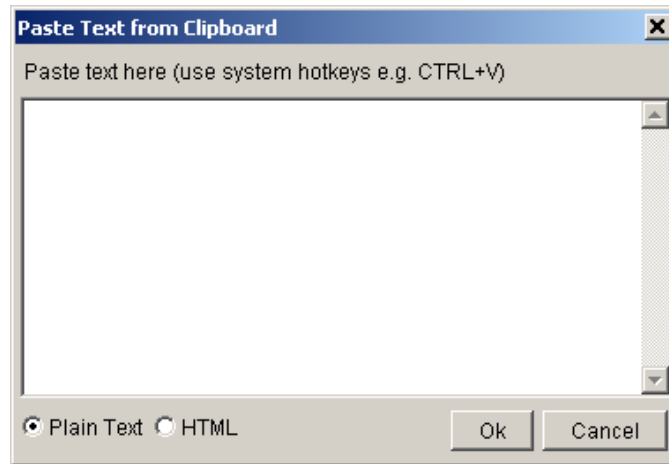
Dialog Screenshot:

Parameter: INSERTSPAN
Syntax: INSERTSPAN;Icon_URL
Default Icon: 
Default Icon URL: icons/insertspan.gif
Description: When clicked, a dialog box will ask for the stylesheet class to apply to the span. The available classes will be the generic classes and span context classes. After pressing OK, a span tag pair will be inserted.





Dialog Screenshot:

Parameter: INSERTTEXT
Syntax: INSERTTEXT;Icon_URL
Default Icon: 
Default Icon URL: icons/inserttext.gif
Description: Inserts a text as "plain text" or HTML at the current cursor position. A dialog box will prompt the user to enter the text. Use the parameter INSERTTEXT_HTML within the configuration file 'config-xml' to activate the HTML radio button.




Dialog Screenshot:

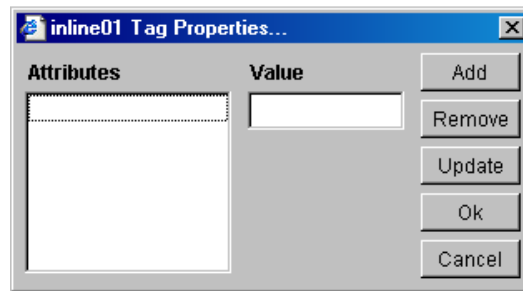
Parameter: **HR**
 Syntax: HR;Icon_URL
 Default Icon: 
 Default Icon URL: icons/hr.gif
 Description: Inserts a horizontal ruler/line at the current cursor position.

Parameter: **SPECIALCHARACTER**
 Syntax: SPECIALCHARACTER;Icons_URL
 Default Icon: 
 Default Icon URL: icons/charmap.gif
 Description: Inserts a special character at the current cursor location. A dialog box will show the characters available. After selecting a character and pressing "insert", the character will be inserted into the document at the current cursor location.




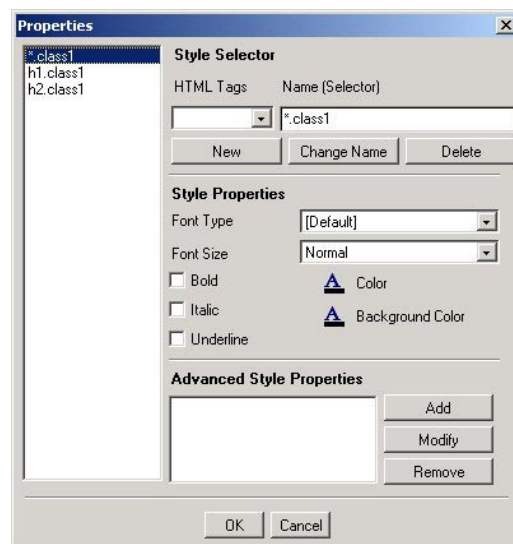
Dialog Screenshot:

Parameter: **SHOWNALLCHAR**
 Parameter: SHOWNALLCHAR;Icon_URL
 Default Icon: 
 Default Icon URL: icons/shownallchar.gif
 Description: When active, control characters like ¶ and unknown tags will be displayed. The unknown tags can be edited by double-clicking on the icon. Any changes within unknown tags will be preserved. The dialog screenshot below shows the "Edit Tag" dialog. It opens after right clicking on a custom/unknown tag and choosing the "Tag properties" pop up menu item.




Dialog Screenshot:

Parameter: **PROPERTIESDIALOG**
 Syntax: PROPERTIESDIALOG;icon_URL
 Default Icon: 
 Default Icon URL: icons/properties.gif
 Description: When TRUE, style sheet properties can be adjusted in a settings dialog.

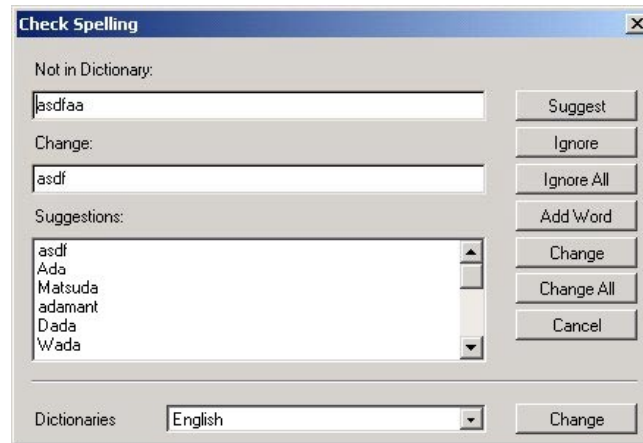


Dialog Screenshot:


Parameter: **SPELLCHECK**
 Syntax: SPELLCHECK;icon_URL
 Default Icon: 
 Default Icon URL: icons/spellcheck.gif
 Description: Performs a spell check of the whole document. Unrecognized words can be added into the Dictionary/Lexicon file. The new word will be recognized next time the spell checking is performed. A feature to change the spell check dictionary language is also provided. Available languages are specified in the "config.xml" file. Please see the chapter Configuration File for further details on the configuration file.


Required:


- The config.xml file's parent element SPELLINGCHECKERS contains the parameter SPELLINGCHECKER for at least one language
- Spelling checker library
- Dictionary/Lexicon files




Dialog Screenshot:

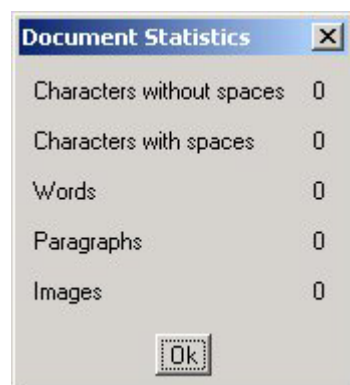
Parameter: **CLEAN**
 Syntax: CLEAN;Icon_URL
 Default Icon: 
 Default Icon URL: icons/clean.gif
 Description: Deletes any inline element tags in the current selection, except for custom and unknown tags.

Parameter: **STANDALONE**
 Syntax: STANDALONE;Icon_URL
 Default Icon: 
 Default Icon URL: icons/standalone.gif
 Description: Toggles to display edit-on Pro in a separate window out of the browser window and back to the browser window.

Parameter: **HELP**
 Syntax: PROPERTIESDIALOG;Icon_URL
 Default Icon: 
 Default Icon URL: icons/help.gif
 Description: When clicked, a new window will be opened to show the help file. The help file URL must be defined in the applet parameter. Please modify/create the help file according to your needs.

Required: HELP parameter in the applet (See “Applet Parameter API”)

Parameter: **DOCSTAT**
 Default Icon: 
 Default Icon URL: icons/docstat.gif
 Description: When clicked a dialog that contains the statistics of the current document will be shown.



Dialog Screenshot:

Parameter: **INFO**

Default Icon: 

Description: When clicked, the about dialog will be displayed with version and copyright information. Additional information about the runtime environment can be obtained by clicking on "System Info". For this parameter, the default icon cannot be changed, and It is not possible to enable/disable the info icon.



Dialog Screenshot:



Parameter: **SEPARATOR**
 Syntax: SEPARATOR
 Description: Separates a group of buttons from others through a vertical line.

2.3.3 Custom Button

There are several types of custom buttons:

1. `urlicon` to open an url in a target frame
 Required information :
 - Parameter name : button identity (also appears as tooltip, see below)
 - icon file
 - url which will be opened when the button is clicked (may be relative to documentbase)
 Optional Information:
 - Enabled state (Default : true)
 - target window (Default : `_blank`)

Example:

```
<urlicon paramname="UPLOADIMAGE">
  <iconfile>icons/java.gif</iconfile>
  <url>http://www.realobjects.com/eopro3/samples/imageupload/imageupload.htm</url>
>
  <target>_upload</target>
</urlicon>
```

2. `dlgicon` to open a custom dialog
 Required information :
 - Parameter name : button identity (also appears as tooltip, see below)
 - icon file
 - class name of dialog which will be opened when the button is clicked
 Optional Information:
 - Enabled state (Default : true)

Example :

```
<dlgicon paramname="CUSTOMDLG1">
  <iconfile>../samples/customdialogs/insertlink.gif</iconfile>
  <classname>com.realobjects.customdialogs.insertlink.InsertLinkDialog</classname>
</dlgicon>
```

3. `jsicon` to execute a JavaScript function within the page which contains the `<applet>` tag.
 Required information:
 - Parameter name : button identity (also appears as tooltip, see below)

- icon file
- name of the JavaScript function to execute
- parameter of the JavaScript function

Optional Information:

- Enabled state (Default : true)

Example :

```
<jsicon paramname="ClearContent" enabled="true">
  <iconfile>icons/new.gif</iconfile>
  <functionname>clearContent</functionname>
  <param>EditorApplet</param>
</jsicon>
```

Note: The JavaScript function must have at least one parameter which refers to the EditorApplet, the parameters listed in the jsicon element will be passed as the second parameter and so on.

To display a tooltip for each custom button, the tooltip string must be defined in the locale files with the same ID as the button parameter name defined in the toolbar xml file. Otherwise the tooltip will not be displayed.

2.3.4 Sample Toolbar Definition File in XML Format

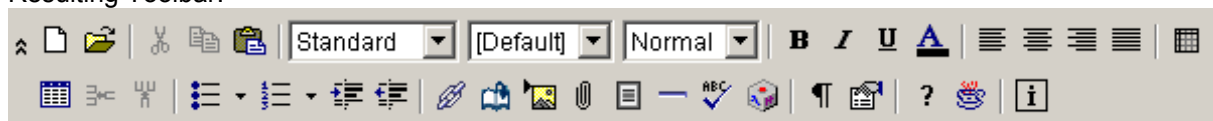
```
<toolbar collapse="false">
  <icon paramname="NEW">
  </icon>
  <icon paramname="LOAD">
  </icon>
  <icon paramname="GETHTMLDATAURL">
    <iconfile>icons/save.gif</iconfile>
  </icon>
  <separator />
  <icon paramname="CUT">
  </icon>
  <icon paramname="COPY">
  </icon>
  <icon paramname="PASTE">
  </icon>
  <separator />
  <choice paramname="PARAGRAPHSTYLE" />
  <choice paramname="FONT" />
  <choice paramname="FONTSIZE" />
  <separator />
  <icon paramname="BOLD">
    <iconfile>icons/bold.gif</iconfile>
  </icon>
  <icon paramname="ITALIC">
    <iconfile>icons/italic.gif</iconfile>
  </icon>
  <icon paramname="UNDERLINE">
  </icon>
  <icon paramname="COLOR">
    <iconfile>icons/autocolor.gif</iconfile>
  </icon>
  <separator />
  <icon paramname="ALIGNLEFT">
    <iconfile>icons/left.gif</iconfile>
  </icon>
  <icon paramname="ALIGNCENTER">
    <iconfile>icons/center.gif</iconfile>
  </icon>
  <icon paramname="ALIGNRIGHT">
    <iconfile>icons/right.gif</iconfile>
  </icon>
  <icon paramname="ALIGNJUSTIFY">
    <iconfile>icons/justify.gif</iconfile>
  </icon>
  <separator />
  <icon paramname="INSERTTABLEDIALOG">
  </icon>
  <icon paramname="INSERTTABLE">
  </icon>
  <icon paramname="INSERTROW">
  </icon>
  <icon paramname="INSERTCOLUMN">
  </icon>
  <separator />
```

```

<icon paramname="UNORDEREDLIST">
  <iconfile>icons/disculist.gif</iconfile>
  <subicon paramname="DISCUNORDEREDLIST">
    <iconfile>icons/disculist.gif</iconfile>
  </subicon>
  <subicon paramname="CIRCLEUNORDEREDLIST">
    <iconfile>icons/circleulist.gif</iconfile>
  </subicon>
  <subicon paramname="SQUAREUNORDEREDLIST">
    <iconfile>icons/squareulist.gif</iconfile>
  </subicon>
</icon>
<icon paramname="ORDEREDLIST">
  <iconfile>icons/numberolist.gif</iconfile>
  <subicon paramname="LOWERALPHAORDEREDLIST">
    <iconfile>icons/lowerolist.gif</iconfile>
  </subicon>
  <subicon paramname="UPPERALPHAORDEREDLIST">
    <iconfile>icons/upperolist.gif</iconfile>
  </subicon>
  <subicon paramname="NUMBEREDORDEREDLIST">
    <iconfile>icons/numberolist.gif</iconfile>
  </subicon>
</icon>
<icon paramname="ALPHABETLIST">
  <iconfile>icons/alist.gif</iconfile>
</icon>
<icon paramname="INCIDENT">
  <iconfile>icons/incident.gif</iconfile>
</icon>
<icon paramname="DECIDENT">
  <iconfile>icons/decident.gif</iconfile>
</icon>
<separator />
<icon paramname="LINK">
  <iconfile>icons/link.gif</iconfile>
</icon>
<icon paramname="BOOKMARK">
  <iconfile>icons/bookmark.gif</iconfile>
</icon>
<icon paramname="IMAGE">
</icon>
<icon paramname="INSERTHTML">
  <iconfile>icons/file.gif</iconfile>
</icon>
<icon paramname="INSERTTEXT">
</icon>
<icon paramname="HR">
</icon>
<icon paramname="SPELLCHECK">
</icon>
<icon paramname="SPECIALCHARACTER">
  <iconfile>icons/charmap.gif</iconfile>
</icon>
<separator />
<icon paramname="SHOWNALLCHAR">
  <iconfile>icons/shownallchar.gif</iconfile>
</icon>
<icon paramname="PROPERTIESDIALOG">
</icon>
<separator />
<icon paramname="HELP">
  <iconfile>icons/help.gif</iconfile>
</icon>
<jsicon paramname="JSBUTTON" enabled="true">
  <iconfile>icons/java.gif</iconfile>
  <functionname>js_icon_preview</functionname>
  <param>EditorApplet</param>
</jsicon>
<icon paramname="INFO">
  <iconfile>icons/info.gif</iconfile>
</icon>
</toolbar>

```

Resulting Toolbar:



2.4 JavaScript API

Note:

- Using the JavaScript API of edit-on Pro requires that the browser/JavaVM combination supports "LiveConnect". "LiveConnect" refers to JavaScript and Java communication, inside a web browser. This is supported e.g. by Internet Explorer on Windows, Netscape on Windows and Linux but, for example, not by Internet Explorer on MacOS.
- Data that gets inserted through the JavaScript API is expected to be already encoded correctly, i.e. URLs are URL encoded and all textual content in attributes and elements is HTML encoded (e.g. & is inserted as &). The only exception is the **insertImage(2)** function: in this function the inserted data is HTML encoded automatically by edit-on Pro.

To use similar functions on Mac OS, an alternate API is provided which can be used on Mac OS X. This API emulates the LiveConnect functionality missing in the Macintosh Java implementation. For details see "JavaScript API for Mac OS".

setHTMLData (1)

Syntax: `void setHTMLData(String strUrlAddress)`

Description: Loads a HTML page from `strUrlAddress` into the editor. The editor will read and parse the page and display it.

Sample: `document.MyEditor.setHTMLData("http://myserver.com/docs/test1.htm");`

setHTMLData (2)

Syntax: `void setHTMLData(String strBaseUrl, String strContents)`

Description: Loads HTML data from `strContents` into the editor. The editor will read and parse the supplied data and display it. `StrBaseUrl` can be used to specify a base URL for images.

Sample: `MyContentString="<p>Paragraph with bold text</p>";
document.MyEditor.setHTMLData("http://", MyContentString);`

getHTMLData

Syntax: `String getHTMLData(String strBaseUrl)`

Description: Retrieves the HTML Data from the editor. When called, it will export the content of the editor as HTML. `StrBaseUrl` can be used to specify a base URL for images.

Sample: `MyResultString = document.MyEditor.getHTMLData("http://");`

postDocumentToServer

Syntax: `void postDocumentToServer()`

Description: Triggers the post mechanism in the applet that will post the document content to the CGI/ASP/PHP/JSP URL specified in the GETHTMLDATAURL applet parameter. The request will be a POST request and the data will be stored in a field with the name "HTMLDATA". See also: "Using direct HTTP connections".

insertHTMLData (1)

Syntax: `void insertHTMLData(String strUrlAddress)`

Description: Inserts a HTML page from `strUrlAddress` into the editor at the cursor position. The editor will read and parse the page and insert it at the cursor position.

Sample: `document.MyEditor.insertHTMLData("http://www.myserver.com/docs/test1.htm");`

insertHTMLData (2)

Syntax: `void insertHTMLData(String strBaseUrl, String strContents)`

Description: Inserts HTML data from `strContents` into the editor at the cursor position. The editor will read and parse the page and insert it at the cursor position. `strBaseUrl` can be used to specify a base URL for images.

Sample: `MyContentString="<p>Paragraph with bold text</p>";
document.MyEditor.insertHTMLData("http://", MyContentString);`

getSelectedHTMLData

Syntax: `String getSelectedHTMLData(String strBaseUrl)`

Description: Retrieves the selected HTML Data from the editor. When called, it will export the content of the editor as HTML. `strBaseUrl` can be used to specify a base URL for images.

Sample: `MyResultString = document.MyEditor.getSelectedHTMLData("http://");`

getPlainText

Syntax: `String getPlainText()`

Description: Retrieves the content (text and links) from the editor. Line breaks “
” and paragraphs “<p></p>” will be transformed into CRLF while unknown tags, images and other HTML tags will be ignored. When called, it will export the content of the editor as plain text format.

Sample: `MyResultString = document.MyEditor.getPlainText();`

insertImage (1)

Syntax: `void insertImage(String strUrlAddress)`

Description: Inserts an image (JPG or GIF) at the current cursor location. The image must be described by the parameter `strUrlAddress`.

Sample: `document.MyEditor.insertImage("http://www.heise.de/icons/ho/heise.gif");`

insertImage (2)

Syntax: `void insertImage(String src, String strWidth, String strHeight, String strBorder, String alt)`

Description: Inserts an image (JPG or GIF) at the current cursor location. The image is described by the parameters `String src`, `String strWidth`, `String strHeight`, `String strBorder` and `String alt`

Sample: `imagesrc="http://www.heise.de/icons/ho/heise.gif"
imagewidth="200"
imageheight="100"
imageborder="3"
imagealt="Your text..."
document.MyEditor.insertImage(imagesrc,imagewidth,imageheight,imageborder,
imagealt);`

clear

Syntax: `void clear()`

Description: Clears the content of the current document within the editor. This has the same effect as pressing the “NEW” button.

Sample: `document.MyEditor.clear();`

insertCustomTag

Syntax: `void insertCustomTag(String strTagName, String strEmpty, String strAttributes)`

Description: Inserts the opening custom tag named `strTagName`, before a selected text within the document. It also inserts the corresponding closing tag after a selected text within the document (if `strEmpty` is set to “false”). An empty tag named `strTagName` is inserted if `strEmpty` is set to “true”. The opening custom tag will have the attributes specified in `strAttributes`. Please note that `strTagName` should only contain the tag’s name, excluding the ‘<’ and ‘>’. `strAttributes` should be formatted in the attribute format of XML tags (`name='value'`)

Sample: `document.MyEditor.insertCustomTag("realobjects", false, "user='developer'");`

hasContentChanged

Syntax: `String hasContentChanged()`

Description: Retrieves the status of the document in the editor. Returns true if any operation has been executed to alter the document and returns false if otherwise.

Sample: `bContentChanged = document.MyEditor.hasContentChanged();`

setStyleSheet

Syntax: `String setStyleSheet(String strStyles)`

Description: Loads styles from `strStyles` into the editor. The editor will read and parse the supplied data, and use this in displaying its contents. Any existing styles inside the document will be overwritten by this action. To add new styles without erasing existing styles, please use `addStyleSheet`. See section 3.7.4 for information about supported style sheets in edit-on Pro.

Sample: `MyStyleString="p {color:blue;}";
document.MyEditor.setStyleSheet("http://", MyStyleString);`

getStyleSheet

Syntax: `String getStyleSheet()`

Description: Retrieves the styles inside the document. Returns a string that contains the styles inside the document.

Sample: `MyStyleSheetString = document.MyEditor.getStyleSheet();`

addStyleSheet

Syntax: `String addStyleSheet(String strStyles)`

Description: Loads styles from `strStyles` into the editor. The editor will read and parse the supplied data, and use it to display its content. Any existing styles inside the document will not be overwritten by this action; the new styles will be combined with the existing ones.

See “Styles Supported by edit-on Pro” for information about supported style sheets in edit-on Pro.

Sample:

```
MyStyleString="p {text-decoration:underline;}";
document.MyEditor.addStyleSheet(MyStyleString);
```

setStyleSheetURL

Syntax:

```
String setStyleSheetURL(String strURLAddress)
```

Description: Loads styles from stylesheet document (.css) resides at `strURLAddress` into the editor. The editor will read and parse the supplied data, and use it to display its content. See “Styles Supported by edit-on Pro” for information about supported style sheets in edit-on Pro.

Sample:

```
document.MyEditor.setStyleSheetURL("http://myserver.com/docs/style.css");
```

cleanStyleSheet

Syntax:

```
void cleanStyleSheet()
```

Description: Erases any existing styles inside the document.

Sample:

```
document.MyEditor.cleanStyleSheet();
```

getCurrentElement

Syntax:

```
String getCurrentElement()
```

Description: Retrieves the tag at the current cursor position, including the textual content as well as all child elements.

Sample:

```
MyResultString = document.MyEditor.getCurrentElement();
```

setCurrentElementContent

Syntax:

```
void setCurrentElementContent(String strContent)
```

Description: Sets the content (inner HTML) of the tag in the current cursor position.

Sample:

```
document.MyEditor.setCurrentElementContent("the new content");
```

getCurrentElementContent

Syntax:

```
String getCurrentElementContent()
```

Description: Retrieves the content (inner HTML) of the tag at the current cursor position.

Sample:

```
MyResultString = document.MyEditor.getCurrentElementContent();
```

set_GETHTMLDATAURL

Syntax:

```
public void set_GETHTMLDATAURL(String strUrl)
```

Description: Changes the value that has been set in the GETHTMLDATAURL parameter of the Applet Parameter API.

Sample:

```
document.MyEditor.set_GETHTMLDATAURL("http://www.realobjects.com");
```

setHeadElementContent

Syntax: `public void setHeadElementContent(String strContent)`

Description: Sets the content of the document's <head>...</head> element. This will have no effect if the BODYONLY applet parameter is set to true. For details, please review the Applet Parameter API.

Sample: `document.MyEditor.setHeadElementContent("<meta http-equiv='Content-type' content='text/html; />");`

getHeadElementContent

Syntax: `public String getHeadElementContent()`

Description: Gets the content of the document's <head>...</head> element. This will have no effect if the BODYONLY applet parameter is set to true. For details, please review the Applet Parameter API.

Sample: `document.MyEditor.getHeadElementContent();`

encode

Syntax: `public String encode(String strDecodedString)`

Description: Encodes the specified string into HTML entities. The function encodes the passed parameter string depending on the encoding which is specified by the applet parameter "ENCODING" (see the chapter "Applet Parameter API" for details). If this parameter is not specified, the string is HTML encoded (codes from 0-127 (except for space)).

Sample: `strDecodedString="&";
strEncodedString=document.MyEditor.encode(strDecodedString); //This will return
"&";`

decode

Syntax: `public String decode(String strEncodedString)`

Description: Decodes the specified HTML entities. The function converts the string containing HTML entities to a string without entities. The HTML entities are converted to their corresponding character data.

Sample: `strEncodedString="&";
strDecodedString=document.MyEditor.decode(strEncodedString); //This will return
"&";`

2.5 JavaScript API for Mac OS

2.5.1 Introduction

The JavaScript API described in the previous chapter, is not available on a Macintosh client due to the fact that the supported web browsers on the Mac platform do not offer LiveConnect. LiveConnect means that you are able to call Java methods (of the edit-on Pro applet), from JavaScript in an HTML page (and vice versa).

RealObjects developed this alternative JavaScript API, in order to support the communication between Java and Javascript on the Mac. For details on how to integrate the API in your application see "Integration of the JavaScript API for Macintosh".

Below are two tables which show the equivalents of the functions of the JavaScript API for Mac OS to the JavaScript API functions explained in the previous chapter. The tables list the functions of the alternative JavaScript API functions available on Mac OS X and Mac OS 9 respectively.

Table of the JavaScript API functions available on Mac OS X:

JavaScript API functions for Non-Macintosh platform	Corresponding JavaScript API functions for the Mac OS X platform
setHTMLData(1)	LCE_setHTMLDataURL
setHTMLData(2)	LCE_setHTMLData
insertHTMLData(1)	LCE_insertHTMLDataURL
insertHTMLData(2)	LCE_insertHTMLData
getHTMLData	LCE_getHTMLData
getSelectedHTMLData	LCE_getSelectedHTMLData
getPlainText	LCE_getPlainText
insertImage(1)	LCE_insertImageURL
insertImage(2)	LCE_insertImage
clear	LCE_clear
insertCustomTag	LCE_insertCustomTag
hasContentChanged	LCE_hasContentChanged
setStyleSheet	LCE_setStyleSheet
addStyleSheet	LCE_addStyleSheet
setStyleSheetURL	LCE_setStyleSheetURL
getStyleSheet	LCE_getStyleSheet
cleanStyleSheet	LCE_cleanStyleSheet
getCurrentElement	LCE_getCurrentElement
getCurrentElementContent	LCE_getCurrentElementContent
setCurrentElementContent	LCE_setCurrentElementContent
set_GETHTMLDATAURL	LCE_set_GETHTMLDATAURL
setHeadElementContent	LCE_setHeadElementContent
getHeadElementContent	LCE_getHeadElementContent
postDocumentToServer	LCE_postDocumentToServer

Table of the JavaScript API function available on Mac OS 9:

JavaScript API functions for Non-Macintosh platform	Corresponding JavaScript API functions for the Mac OS 9 platform
postDocumentToServer	LCE_postDocumentToServer

Please note that some major differences exist between this (Macintosh platform-compatible) JavaScript API, and the JavaScript API of the previous chapter. One difference is that the JavaScript API for Mac *can not* be called by means of the JavaScript DOM. It has to be called directly e.g.

```
LCE_setHTMLDataURL("myEditor", "http://www.myServer.com", "myEventHandler");
```

In order to set the HTML data, into edit-on Pro within your web page. Instead of accessing the editor itself via DOM (like "document.editor.setHTMLData" in JavaScript), you must provide every function of this API for Mac with the name that has been given to edit-on Pro in the HTML page containing it. In the case above your applet tag would look like this:

```
<applet name="myEditor" ... >
...
</applet>
```

The URL in the example above specifies the location from which the content is to be set into edit-on Pro.

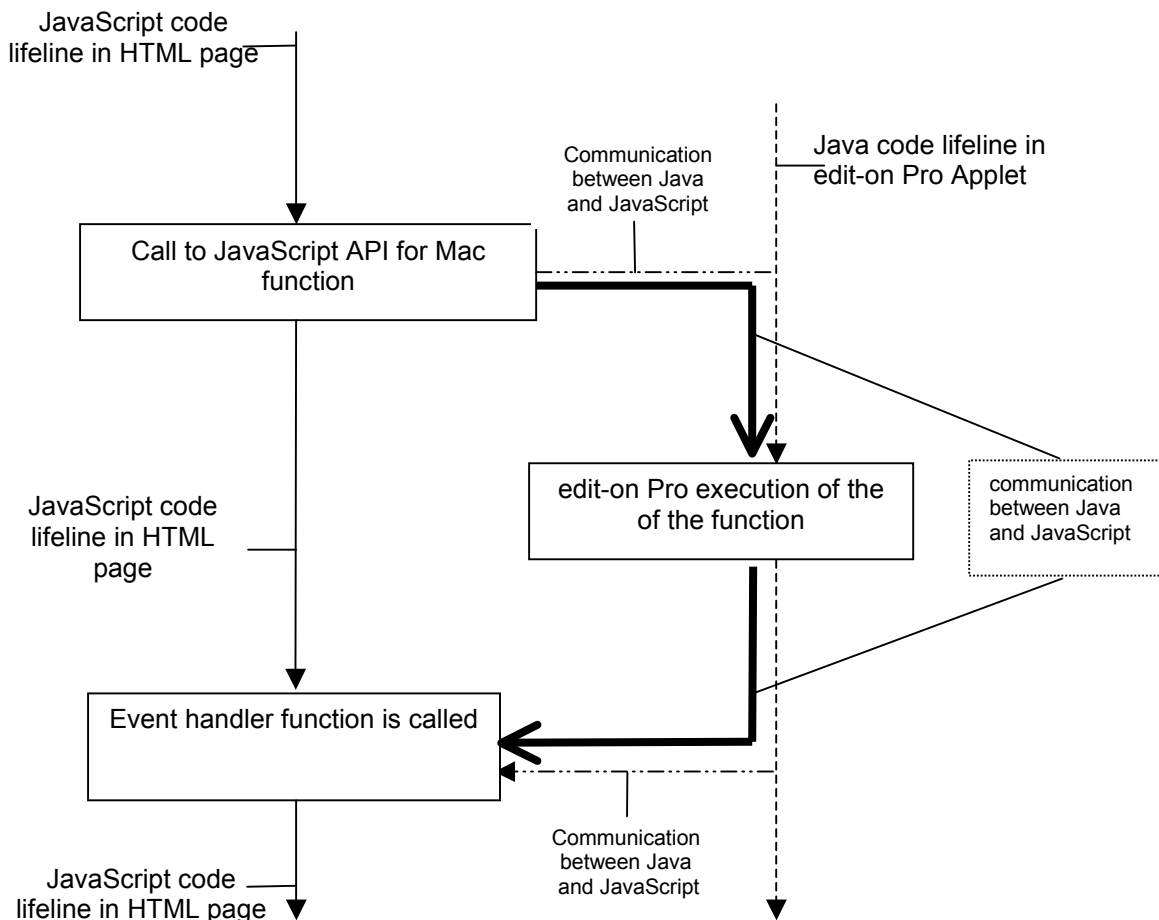
The last parameter is an event handler, which is called after the function has performed the intended action. In the above case, this means that the function "myEventHandler" is called when the data specified by the URL has been set into the editor. In case of a function that receives data from the edit-on Pro applet the data will be passed to the function as parameter of type String. Here's an example for the LCE_getHTMLData function:

```
LCE_getHTMLData("myEditor", "http:// ", "myEventHandler");

// ... other JavaScript code

function myEventHandler( result ) {
    // now access the result from the applet
    // and do something with it
    document.myForm.hiddenField = result;
}
```

The following figure describes this process.



As the figure shows, there is a time between the "Call to JavaScript API for Mac function" box, and the box where "Event handler function is called", where a synchronous behaviour can not be ensured. Only when the "Event handler function is called" can the user rely on the result of the operation, either while getting data from edit-on Pro, or sending data to edit-on Pro.

So, be sure to place any JavaScript code that depends on the result of this API function in the according event handler function. For each function you must have an own event handler. It is even possible to use a different event handler for the same function, if the function is used several times in the same HTML page.

The next section describes the API in more detail.

2.5.2 Restrictions and technical constraints of the JavaScript API for Mac OS

The following restrictions are to consider when using the JavaScript API for Mac OS on **OS 9**:

- The only LCE method which is supported on **Mac OS 9** is LCE_postDocumentToServer. This method may only be called *once* during the lifecycle of an applet instance, so the applet has to be destroyed before it is called again (navigate to a different page). Also, it must be used together with GETHTMLDATAURL and GET_OK_URL and GET_OK_TARGET to ensure proper synchronisation. For details see the chapter "Applet Parameter API".

- There are also certain client requirements which **must** be met to ensure a proper runtime behaviour on Mac OS 9. Details on these constraints and requirements are clearly outlined in the README.TXT included in this release package.
- When working on **Mac OS 9** with **Internet Explorer** please be sure to use **edit-on-pro-signed-macos9.jar** in the archive attribute of the applet tag in your web page. Otherwise the JavaScript API for Mac won't work on OS 9. It is very important that **edit-on-pro-signed-macos9.jar** file is **only used on a Mac OS 9** operating system. For OS X the **edit-on-pro-signed.jar** must be used which is the officially recommended one all platforms but Mac OS 9. As an example how to distinguish between OS 9 and OS X please refer to e.g. the "General API Sample" or the "Bookreview Sample" in the samples folder of the edit-on Pro distribution package.

The constraints above do not apply to **Mac OS X**. All functions of the LCE API are fully supported on this platform. However, this API can only be used on a client running **Mac OS** when working with the **Microsoft Internet Explorer for Mac**. Otherwise the API will not work as intended.

The following restrictions are to consider when using the JavaScript API for Mac OS on **OS X**:

- For technical reasons, the Mac OS JavaScript API does not work when the permanent caching of the editor is activated. To get the Mac OS JavaScript API to work on Mac OS, you need to remove the following parameters from the applet tag:

```
<param name="cache_archive" value="edit-on-pro-signed.jar">
<param name="cache_version" value="...">
<param name="cache_option" value="plugin">
```

Please note those parameters only need to be disabled on Mac OS X. In general they can be used on all other platforms. After removing those parameters, clear your browser cache, then reload the applet. Now the Mac OS X JavaScript API should work properly.

The following restrictions are to consider when using the JavaScript API for Mac OS on both **OS 9 and OS X**:

- The JavaScript API for Mac OS in general requires that the page containing the applet has a unique URL like

```
http://www.myserver.com/eopro/editor.php?id=3ab34cf93
```

where `id=3ab34cf93` is a unique identifier for this URL. This unique id identifier does not necessarily have to look like in this example. It is important that window containing the applet can be distinguished from any other window through the URL. Otherwise it is not guaranteed that the JavaScript API for Mac OS will work properly.

- Make sure that data that gets inserted through the JavaScript API is expected to be already encoded correctly, i.e. URLs are URL encoded and all textual content in attributes and elements is XHTML encoded (e.g. `&` is inserted as `&`). The only exception is **insertImage(2)** function: in this function the inserted data is encoded automatically by edit-on Pro.

2.5.3 API Description for Macintosh platform

LCE_setHTMLDataURL

Syntax: `void LCE_setHTMLDataURL(String appletName, String strUrlAddress, String eventHandler)`

Description: Loads a HTML page from `strUrlAddress` into the editor specified by `appletName`. The editor will read and parse the page and display it. The event handler will be called after the HTML data was loaded.

Sample:

```
LCE_setHTMLDataURL("MyEditor", "http://myserver.com/docs/test1.htm",
"setHTMLDataURLFinished");

function setHTMLDataURLFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_setHTMLData

Syntax: void LCE_setHTMLData(String appletName, String strBaseUrl, String strContents, String eventHandler)

Description: Loads HTML data from strContents into the editor specified by appletName. The editor will read and parse the supplied data and display it. strBaseUrl can be used to specify a base URL for images. The event handler will be called after the HTML data was loaded.

Sample:

```
MyContentString="<p>Paragraph with <strong>bold</strong> text</p>";
LCE_setHTMLData("MyEditor" , "http://", MyContentString,
"setHTMLDataFinished");

function setHTMLDataFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_getHTMLData

Syntax: void LCE_getHTMLData(String appletName, String strBaseUrl, String eventHandler)

Description: Retrieves the HTML Data from the editor specified by appletName. When called, it will export the content of the editor into HTML format. StrBaseUrl can be used to specify a base URL for images. Instead of immediately returning the result as String, an event handler function is called when the operation is finished. The result will be passed to the event handler function as parameter of type *String*.

Sample:

```
LCE_getHTMLData("MyEditor", "http://", "getHTMLDataFinished");

function getHTMLDataFinished ( result ) {
    myResultString = result;
}
```

LCE_insertHTMLDataURL

Syntax: void LCE_insertHTMLDataURL(String appletName, String strUrlAddress, String eventHandler)

Description: Inserts a HTML page from strUrlAddress into the editor specified by appletName at the cursor position. The editor will read and parse the page and insert it at the cursor position. The event handler will be called after the HTML data has been inserted.

Sample:

```
LCE_insertHTMLDataURL("MyEditor" , "http://www.myserver.com/docs/test1.htm",
"insertHTMLDataURLFinished" );

function insertHTMLDataURLFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_insertHTMLData

Syntax: void LCE_insertHTMLData(String appletName, String strBaseUrl, String strContents, String eventHandler)

Description: Inserts HTML data from strContents into the editor specified by appletName at the cursor position. The editor will read and parse the page and insert it at the cursor position. strBaseUrl can be used to specify a base URL for images. The event handler will be called after the HTML data has been inserted.

Sample:

```
MyContentString="<p>Paragraph with <strong>bold</strong> text</p>";
insertHTMLData("MyEditor", "http://", MyContentString,
"insertHTMLDataFinished");

function insertHTMLDataFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_getSelectedHTMLData

Syntax: void LCE_getSelectedHTMLData(String appletName, String strBaseUrl, String eventHandler)

Description: Retrieves the selected HTML Data from the editor specified by appletName. When called, it will export the content of the editor into HTML format. StrBaseUrl can be used to specify a base URL for images. Instead of immediately returning the result as String, an event handler function is called when the operation is finished. The result will be passed to the event handler function as parameter of type *String*.

Sample:

```
LCE_getSelectedHTMLData("MyEditor", "http://", "getSelectedHTMLDataFinished");

function getSelectedHTMLDataFinished ( result ) {
    MyResultString = result;
}
```

LCE_getPlainText

Syntax: void LCE_getPlainText(String appletName, String eventHandler)

Description: Retrieves the content (text and links) from the editor specified by appletName. Line breaks "
" and paragraphs "<p></p>" will be transformed into CRLF while unknown tags, images and other HTML tags will be ignored. When called, it will export the content of the editor into a plain text format. Instead of immediately returning the result as String, an event handler function is called when the operation is finished. The result will be passed to the event handler function as parameter of type *String*.

Sample:

```
LCE_getPlainText("MyEditor", "getPlainTextFinished");

function getPlainTextFinished ( result ) {
    MyResultString = result;
}
```

LCE_insertImageURL

Syntax: void LCE_insertImageURL(String appletName, String strUrlAddress, String eventHandler)

Description: Inserts an image (JPG or GIF) at the current cursor location. The image must be described by the parameter strUrlAddress. The editor is specified by appletName.

Sample:

```
LCE_insertImageURL("MyEditor", "http://www.heise.de/icons/ho/heise.gif",
"insertImageURLFinished" );

function insertImageURLFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_insertImage

Syntax: void LCE_insertImage(String appletName, String src, String strWidth, String strHeight, String strBorder, String alt, String eventHandler)

Description: Inserts an image (JPG or GIF) at the current cursor location. The image is described by the parameter String src, String strWidth, String strHeight, String strBorder and String alt. The editor is specified by appletName.

Sample:

```
imagesrc="http://www.heise.de/icons/ho/heise.gif"
imagewidth="200"
```

```

imageheight="100"
imageborder="3"
imagealt="Your text..."

LCE_insertImage("MyEditor", imagesrc, imagewidth, imageheight, imageborder,
imagealt, "insertImageFinished");

function insertImageFinished () {
    // go on with the Javascript code you want to execute
}

```

LCE_clear

Syntax: void LCE_clear(String appletName, String eventHandler)

Description: Clears (deletes) the content of the current document within the editor. It has the same function as pressing the "NEW" button.

Sample:

```

LCE_clear("MyEditor", "clearFinished");

function clearFinished () {
    // go on with the Javascript code you want to execute
}

```

LCE_insertCustomTag

Syntax: void LCE_insertCustomTag(String appletName, String strTagName, String strEmpty, String strAttributes, String eventHandler)

Description: Inserts the opening custom XML tag named strTagName before a selected text of the document and the corresponding closing tag after a selected text of the document if strEmpty is set to false and inserts an empty tag named strTagName if strEmpty is set to true. The opening custom XML tag will have the attributes specified in strAttributes. Please note that the strTagName should only contain the tag's name, excluding the '<' and '>'. strAttributes should be formatted in the attribute format of XML tags (name='value').

Sample:

```

LCE_insertCustomTag("MyEditor", "realobjects", false, "user='developer'",
"insertCustomTagFinished");

function insertCustomTagFinished () {
    // go on with the Javascript code you want to execute
}

```

LCE_hasContentChanged

Syntax: void LCE_hasContentChanged(String appletName, String eventHandler)

Description: Retrieves the status of the document in the editor specified by appletName. Returns true if any operation has been executed to alter the document, and returns false if otherwise. Instead of returning the result immediately, after the operation has finished, an event handler function will be called. The result of the operation will be passed to the event handler function as "true" or "false".

Sample:

```

LCE_hasContentChanged("MyEditor", "hasContentChangedFinished");

function hasContentChangedFinished ( result ) {
    bContentChanged = result;
}

```

LCE_setStyleSheet

Syntax: void LCE_setStyleSheet(String appletName, String strStyles, String eventHandler)

Description: Loads styles from strStyles into the editor specified by appletName. The editor will read and parse the supplied data, and use it to display the editors contents. Any

existing styles inside the document will be overwritten by this action. To add a style without erasing existing styles, please use `addStyleSheet`.

Sample:

```
MyStyleString="p {color:blue;}";
LCE_setStyleSheet("MyEditor", "http://", MyStyleString,
"setStyleSheetFinished");

function setStyleSheetFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_getStyleSheet

Syntax: `void LCE_getStyleSheet(String appletName, String eventHandler)`

Description: Retrieves the styles inside the document, and returns a string that contains the styles inside the document. Instead of returning the result immediately, after the operation has finished, an event handler function will be called. The result of the operation will be passed to the event handler function as type *String*.

Sample:

```
LCE_getStyleSheet("MyEditor", "getStyleSheetFinished");

function getStyleSheetFinished ( result ) {
    MyResultString = result;
}
```

LCE_addStyleSheet

Syntax: `void LCE_addStyleSheet(String appletName, String strStyles, String eventHandler)`

Description: Loads styles from `strStyles` into the editor specified by `appletName`. The editor will read and parse the supplied data, and use it to display the editor's contents. Any existing styles inside the document will not be overwritten by this action; the new styles will be combined with the existing styles.

Sample:

```
MyStyleString="p {text-decoration:underline;}";
LCE_addStyleSheet("MyEditor", MyStyleString, "addStyleSheetFinished");

function addStyleSheetFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_setStyleSheetURL

Syntax: `void LCE_setStyleSheetURL(String appletName, String strURLAddress, String eventHandler)`

Description: Loads styles from the stylesheet document (.css) which resides at `strUrlAddress`, into the editor specified by `appletName`. The editor will read and parse the supplied data and use it to display the editor content.

Sample:

```
LCE_setStyleSheetURL("MyEditor", "http://myserver.com/docs/style.css",
"setStyleSheetURLFinished");

function setStyleSheetURLFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_cleanStyleSheet

Syntax: `void LCE_cleanStyleSheet(String appletName, String eventHandler)`

Description: Erases any existing styles inside the document of the editor specified by `appletName`.

Sample:

```
LCE_cleanStyleSheet("MyEditor", "cleanStyleSheetFinished");
```

```
function cleanStyleSheetFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_getCurrentElement

Syntax: void LCE_getCurrentElement(String appletName, String eventHandler)

Description: Retrieves the tag at the current cursor position including the textual content as well as all HTML or custom/unknown tags. After the operation has finished, an event handler will be called. The result of the operation will be passed to the event handler as a parameter of type *String*.

Sample:

```
LCE_getCurrentElement("MyEditor", "getCurrentElementFinished");

function getCurrentElementFinished( result ) {
    MyResultString = result;
}
```

LCE_setCurrentElementContent

Syntax: void LCE_setCurrentElementContent(String appletName, String strContent, String eventHandler)

Description: Sets the content (inner HTML) of the tag at the current cursor position. After the function is executed in edit-on Pro an event handler is called.

Sample:

```
LCE_setCurrentElementContent("MyEditor" , "the new content",
"setCurrentElementContentFinished");

function setCurrentElementContentFinished () {
    // go on with the Javascript code you want to execute
}
```

LCE_getCurrentElementContent

Syntax: void LCE_getCurrentElementContent(String appletName, String eventHandler)

Description: Retrieves the content (inner HTML) of the tag at the current cursor position. After the operation has finished, an event handler will be called. The result of the operation will be passed to the event handler as parameter of type *String*.

Sample:

```
LCE_getCurrentElementContent("MyEditor", "getCurrentElementContentFinished");

function getCurrentElementContentFinished ( result ) {
    MyResultString = result;
}
```

LCE_set_GETHTMLDATAURL

Syntax: public void LCE_set_GETHTMLDATAURL(String appletName, String strUrl, String eventHandler)

Description: Changes the value that has been set in the GETHTMLDATAURL parameter of the Applet Parameter API.

Sample:

```
LCE_set_GETHTMLDATAURL("MyEditor" , "the new GETHTMLDATAURL value",
"set_GETHTMLDATAURLFinished");

function set_GETHTMLDATAURLFinished() {
    // go on with the Javascript code you want to execute
}
```

LCE_setHeadElementContent

Syntax: public void LCE_setHeadElementContent(String appletName, String strContent, String eventHandler)

Description: Sets the content of the document's <head>...</head> element. This will have no effect if the BODYONLY applet parameter is set to true. For details, please review the Applet Parameter API.

Sample:

```
LCE_setHeadElementContent("MyEditor" , "the new content of the <head>...</head>
element" , "setHeadElementContentFinished");
```

```
function setHeadElementContentFinished() {
    // go on with the Javascript code you want to execute
}
```

LCE_getHeadElementContent

Syntax: `public String LCE_getHeadElementContent(String appletName, String eventHandler)`

Description: Gets the content of the document's <head>...</head> element. This will have no effect if the BODYONLY applet parameter is set to true. For details, please review the Applet Parameter API.

Sample:

```
LCE_getHeadElementContent("MyEditor" , "getHeadElementContentFinished");
```

```
function getHeadElementContentFinished(result) {
    MyResultString = result;
}
```

LCE_postDocumentToServer

Syntax: `public void LCE_postDocumentToServer(String appletName)`

Description: Triggers the post mechanism in the applet that will post the document content to the the CGI/ASP/PHP/JSP URL specified in the GETHTMLDATAURL applet parameter. The request will be a POST request and the data will be stored in a field with the name "HTMLDATA". See also: "Using direct HTTP connections".

NOTE: This function doesn't provide an event handler. It uses the event handler ONDATAPOSTED as described in the applet parameter API description.

Sample:

```
LCE_postDocumentToServer ("MyEditor");
```

2.5.4 Integration of the JavaScript API for Macintosh

In order to use the JavaScript API for Macintosh for users working with Microsoft Internet Explorer, you will have to include the "macos.js" JavaScript source file into the HTML page where you want to use the API functions.

This normally happens within the <head> element of an HTML page:

```
<html>
<head>
  <script src="macos.js"></script>

  <script type="text/JavaScript">
    // your JavaScript code here...
    // ...
  </script>

  <!-- Rest of the HTML page ... -->
```

The file "macos.js" can be found in the "eopro" folder after unzipping the installation package.

In this folder, there is also an equivalent index page, for use with Macintosh client machines called "index_mac.htm"; this index page demonstrates the use of the Macintosh JavaScript API.

Additionally you have to insert the API parameter "LCE" and set it to "MACIE" in the applet tag:

```
<applet name="myEditor">
  ...
  <param name="LCE" value="MACIE"/>
```

```

    ...
</applet>

```

The "LCE" parameter is also described in the "Applet Parameter API" section.

2.5.5 How to use the JavaScript API for Macintosh when submitting a form

As the behaviour of this API is partly asynchronous, problems could occur when retrieving the editor's content, and then immediately submitting it within an HTML form via the HTML onSubmit event.

Normally, there would be a JavaScript function handling this event. In this function, the editor's content is taken and displayed in a preview window:

```

<form name="myForm" action="preview.asp" method="post"
onSubmit="return myFormOnSubmit();" >
    <input type="hidden" name="HTMLText" >
    <input type="submit" value="Preview" >
    ...
</form>

```

The event handling JavaScript function, for a non-Mac client machine where LiveConnect works, could look like this:

```

function myFormOnSubmit() {
    document.myForm.HTMLText.value = document.myEditor.getHTMLData("http://");
}

```

"myEditor" is the name of the edit-on Pro applet in the <applet> tag, and HTMLTEXT is a hidden input field which will contain the editor's content.

When the form is submitted by pressing the "Preview" button, a server side script ("preview.asp" in the example), can read the HTMLText value of the editor's content at the time of its submission.

With the JavaScript API for Macintosh, the submit action has to be changed, due to the asynchronous behaviour of the getHTMLData equivalent, and of course in regards to the changed functions for the Mac related JavaScript API.

If done with the onSubmit event in the form tag, the editor's content would most likely not be within the hidden input field, at the time the server side script reads it.

A way to overcome this would be to leave the onSubmit attribute out of the form tag:

```

<form name="myForm" action="preview.asp" method="post">

```

and change the Preview button from type "submit" to type "button", so that the form looks like this:

```

<form name="myForm" action="preview.asp" method="post">
    <input type="hidden" name="HTMLText" >
    <input type="button" value="Preview" onClick="return myFormOnSubmit();" >
    ...
</form>

```

Please note that the Preview button now has an onClick event handler. The function above handled the onSubmit action.

The function myFormOnSubmit has also to be changed:

```

function myFormOnSubmit() {
    LCE_getHTMLData("editor", "http://", "onGetHTMLDataFinished");
}

```

Please keep in mind that each function has an event handler, which in the case of LCE_getHTMLData, is called onGetHTMLDataFinished. It has to be specified when calling LCE_getHTMLData (...).

Here is the place where the remaining action - filling the hidden input field, and submitting the form - should take place:

```
function onGetHTMLDataFinished( result ) {
    document.myForm.HTMLText.value = result;
    document.myForm.submit();
}
```

The submitting of the form is now done manually with JavaScript.

2.6 Style Sheets

edit-on Pro supports simple style sheets which can be imported, displayed and exported.

2.6.1 Editable Styles and Read Only Styles

Editable styles are styles that can be edited within the editor (see “Style Sheets Support”). These styles can then be exported from edit-on Pro. The editable styles are:

- Styles that are imported into edit-on Pro using the `setStyleSheet` JavaScript API function
- Styles that are imported into edit-on Pro using the `<style>` tag in the HTML Head tag
- Styles that are created within edit-on Pro using the Style Properties Dialog. See “Style Properties Dialog” for information on how to create a new style in edit-on Pro

Read only styles are styles which cannot be edited within the editor. These styles will be taken into account when rendering the document, but they cannot be edited and they will not be exported. The read only styles are:

- Styles that are imported into edit-on Pro using the `STYLESHEETURL` parameter
- Styles that are imported into edit-on Pro using the `SetStyleSheetURL` JavaScript API function
- Styles that are imported into edit-on Pro using the `<link>` tag
- Styles that are imported into edit-on Pro using the `@import` directive in the style sheet section of the HTML Head Tag

2.6.2 Importing Style Sheets

Style sheets can be imported into the document in anyone of the following ways:

- Using the `STYLESHEETURL` parameter
Example:

```
<APPLET>
...
<PARAM NAME = "STYLESHEETURL" VALUE=" " >
...
</APPLET>
```

Set the parameter's value to the URL address where the style sheet is available.
For more information see “Applet Parameter API”

- Using the `SetStyleSheetURL` JavaScript API function
Example:

```
<SCRIPT LANGUAGE="javascript">
    function SetStyleSheetURL ()
    {
        document.editor.SetStyleSheetURL(document.scriptForm.urladdress.value)
    }
</SCRIPT>
```

For more information see “JavaScript API”.

- Using the <link> tag
Example:

```
<HEAD>
    <link rel="stylesheet" type="text/css" media="screen" href="mystyle.css">
    ...
</HEAD>
```

- Using the <style> tag
Example:

```
<HEAD>
    ...
    <style>
        <!--
            a { font-weight: bold; }
        -->
    </style>
    ...
</HEAD>
```

- Using the @import directive in the style sheet section of the HTML Head tag
Example:

```
<HEAD>
    ...
    <style>
        <!--
            { @import ("mystyle.css"); }
        -->
    </style>
    ...
</HEAD>
```

- Using the SetStyleSheet JavaScript API function:
Example:

```
<SCRIPT LANGUAGE="javascript">
    function SetStyleSheet ()
    {
        document.editor.SetStyleSheet (document.scriptForm.urladdress.value)
    }
</SCRIPT>
```

For more information see “JavaScript API”.

2.6.3 Exporting Style Sheets

Style sheets can be exported from the document in one of the following ways:

- Using the GetStyleSheet JavaScript API function:
Example:

```
<SCRIPT LANGUAGE="javascript">
    function GetStyleSheet ()
    {
        document.editor.GetStyleSheet (document.scriptForm.urladdress.value)
    }
</SCRIPT>
```

For more information see “JavaScript API”.

- Using the STYLESHEETDATA field in the POST request
The Style Sheet can be exported using a field called “STYLESHEETDATA” in the POST request. See also: “Using direct HTTP connections”
- The Style Sheet can be exported in the HTML Head section

Please note that only editable styles are exported from edit-on Pro. Read Only styles will not be exported but will be taken into account when the document is rendered.

When the BODYONLY applet parameter is set to true, the style sheet can be exported using:

- The GetStyleSheet JavaScript API function in the HTML Script Tag
- The STYLESHEETDATA field in the POST request

When the BODYONLY applet parameter is set to false, the style sheet can be exported using:

- The GetStyleSheet JavaScript API function in the HTML Script Tag
- The STYLESHEETDATA field in the POST request
- The HTML Head section

When LiveConnect is not available the style sheet can be exported using:

- The STYLESHEETDATA field in the POST request
- The HTML Head section (when BODYONLY is set to false).

2.6.4 Styles Supported by edit-on Pro

The following styles properties are currently supported by edit-on Pro:

2.6.4.1 Font Properties

- **Font Family**
 - Syntax : font-family : <family-name>
 - Possible values : any font-family name
 - Initial Value : determined by the java virtual machine
- **Font Style**
 - Syntax : font-style : <value>
 - Possible values : normal | italic
 - Initial value : normal
- **Font Weight**
 - Syntax : font-weight : <value>
 - Possible values : normal | bold
 - Initial Value : normal
- **Font Size**
 - Syntax : font-size : <absolute-size> | <length>
 - Possible values :
 - <absolute-size> : xx-small | x-small | small | medium | large | x-large | xx-large
 - <length> : --pt, pc, px, in, cm, mm (e.g. 12pt)
 - Initial value : medium (16pt)

2.6.4.2 Color and Background Properties

- **Color**

Syntax : color : <color>

Initial Value : black

- **Background Color**

Syntax : background-color : <value>

Possible values : <color> | transparent

Initial Value : transparent

Possible elements : body, table, td, tr

- **Background**

Syntax : background : <value>

Possible values : <color> | transparent

Initial Value : transparent

Possible elements : body, table, td, tr

2.6.4.3 Text Properties

- **Text Decoration**

Syntax : text-decoration : <value>

Possible values : none | underline

Initial value : none

- **Vertical Alignment**

Syntax : vertical-align : <value>

Possible values : sub | super | baseline

Initial value : baseline

- **Text Alignment**

Syntax : text-align : <value>

Possible values : left | right | center

Initial value : left

2.6.4.4 Box Properties

- **Top Margin**

Syntax : margin-top : <value>

Possible values :

- <length> : pt, pc, px, in, cm, mm

Initial value : 0

- **Bottom Margin**

Syntax : margin-bottom : <value>

Possible values :

- <length> : pt, pc, px, in, cm, mm

Initial value : 21

2.6.4.5 Classification Properties

- **List Style Type**

Syntax : list-style-type : <value>

Possible values : disc | circle | square | decimal | lower-alpha | upper-alpha

2.6.4.6 Units

- **Length Units**

Absolute length units : in, cm, mm, pt, pc

Relative length units : px

- **Color Units**

Predefined color : aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.

RGB colors :

- #rrggbb (e.g. #00cc00)
- #rgb (e.g. #0c0)
- rgb(x,x,x) where x is an integer between 0 and 255 inclusive (e.g. rgb(0,204,0))


2.7 Custom XML Tags and Unknown Tags

2.7.1 Custom XML Tags

Custom XML Tags are tags that are not supported XHTML tags and that are defined in edit-on Pro's configuration file. The tags can have their own rendering icons and custom dialogs. They can be protected from being edited using readonlyarea and readonlytag schemes. They can be specified either as block level or inline level elements, and may be empty or non-empty tags.

Please note any XML tags can be inserted using the "Custom XML Tags" feature.

Block-level elements typically contain inline elements and other block-level elements. When rendered visually, block-level elements usually begin on a new line. Inline elements typically may only contain text and other inline elements. When rendered visually, inline elements do not usually begin on a new line.

Custom tags may have the `alwaysshow` option which allows the editor to display the tag, whether or not the ‘ShownAllChar’ mode (the  toolbar button) is enabled. When the `alwaysshow` attribute is set to true, the tag will always be displayed regardless of the ‘ShownAllChar’ mode. When it is set to false, the tag will only be displayed when the ‘ShownAllChar’ mode is enabled.


Please refer to “Configuration File” for more details on how to specify custom tags.

Custom tags defined in the “config.xml” file, can also be inserted using the “Insert Custom Tag” dialog. The attributes of the custom tags can also be specified within this dialog:



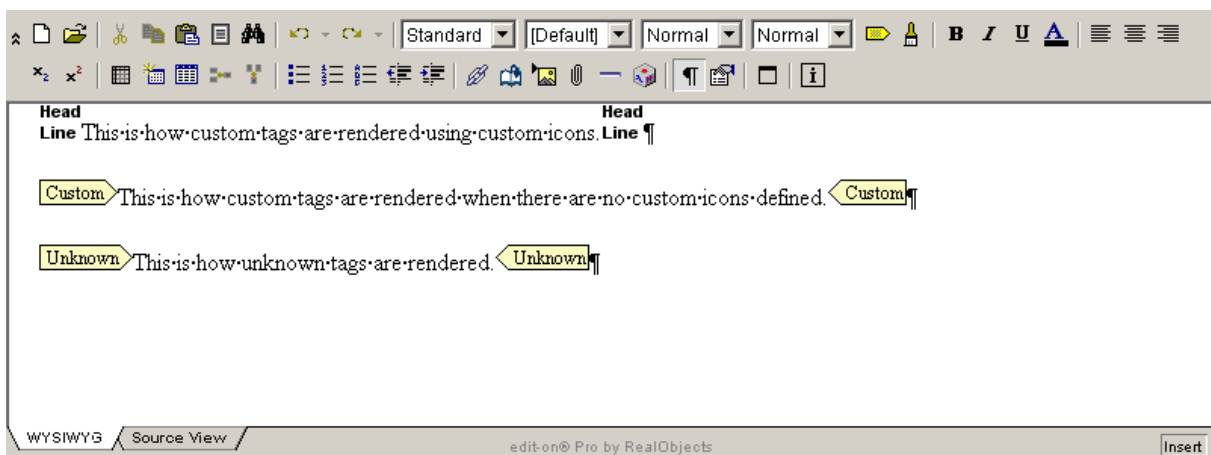
2.7.2 Unknown Tags

Unknown tags are tags which are not valid XHTML tags and are not registered in the configuration file. These tags will be rendered using yellow marks in the WYSIWYG view, and will always be treated as inline elements.

Unknown HTML tags will be displayed within the WYSIWYG view, after pressing the  toolbar button. A “unknown tag” property dialog will appear when clicking the right mouse button on an unknown tag. The content of the unknown tag can be edited while changes will be preserved.

Unknown tags are only displayed when the “ShownAllChar” mode is enabled (the  toolbar button is activated).

Following, is a screenshot showing how custom tags and unknown tags are rendered in edit-on Pro.

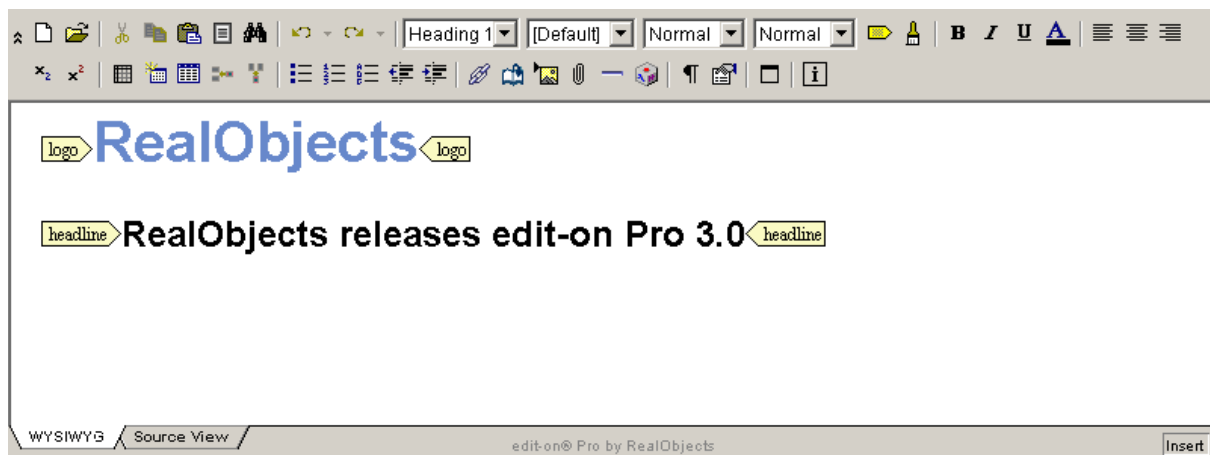


For more information about how to display unknown tags, see “Displaying Unknown Tags”

2.8 Read Only Areas and Read Only Tags

Read Only Areas are parts of the document that are protected. Read Only Areas are marked by a pair of custom tags that are specified as read only area in the configuration file. This particular part can not be edited by the user. Any editing actions, such as typing, deleting, changing the fonts, etc., in the read only area, or in the selection that overlaps with the read only area, will be ignored. When a tag is specified as Read Only Area, it will automatically be a Read Only Tag. Read Only Areas are specifically provided to protect certain parts of the document where editing is not allowed, e.g. company logos, etc..

Read Only Tags are tags that are protected and can not be deleted. Read Only Tags are specified in the configuration file. The user can not delete Read Only Tags but can edit the part of the document marked by Read Only Tags. Read Only Tags may come useful when the user is prohibited to delete certain elements of the document but allowed to alter it, such as the news headline part in a press release.



For more information on how to specify read only areas and read only tags, please refer to "Configuration File".

2.9 Integrating the Clean Up Process

Documents have to be wellformed in order to be displayed within edit-on Pro. If the document is not wellformed, it must first undergo the clean-up process. A document is considered wellformed when it

meets the following criteria: comments are not valid within a tag, comments may not contain two hyphens in a row (other than the beginning and end of the comment), tags must have an end tag (or be closed within the singleton tag itself, for example `<hr />`), all attributes of tags must be quoted, and finally, every XML document must contain one element that completely contains all the other elements.

To ensure the wellformedness of documents imported into edit-on Pro, please integrate a clean up process into your system. edit-on Pro uses the clean up process, either client-side and server side, only if its internal wellformedness checking reports an error during parsing. The applet's internal wellformedness checking process checks for any overlapping tags and/or unclosed/unopened tags.

The clean up process will be invoked by:

- Loading a document using the load icon in the toolbar
- Using the SETHTMLDATAURL in the applet parameter
- Calling setHTMLData and insertHTMLData javascript functions
- Switching from Source View to WYSIWYG View

- Post (save) document when source view mode is active
- Calling the getHTMLData function when the source view mode is active

In case of Custom XML Tags, these tags must be registered in the clean up process engine to allow for the clean up process to correct the document. If the custom XML tags are not registered, then the clean up process will fail, returning an error message to edit-on Pro, refusing to load the document.

There are two options to enable the clean up process: Client Side clean up process, and Server Side clean up process. Both can be enabled by using the cleanupprocess tag in configuration file (e.g : config.xml) .

2.9.1 The Client Side Clean Up Process

The Client Side clean up process can be enabled in the following ways:

- a. Set the class attribute in the CLEANUPPROCESS tag to the client-side clean up process class (if not specified, then edit-on Pro will use tidy.jar and/or tidy.cab). Please make sure that the client-side clean up class implements com.realobjects.eop.applet.custom.CleanUpProcessWrapper.java. This interface has two methods:

- `cleanUp(String strRawData)`
Cleans up the given raw data and returns the cleaned data if no exception occurs.
- `cleanUpClipboard(String strRawData)`
Cleans the raw data inside the clipboard and returns the cleaned data if no exception occurs. This method could be used if it is needed to differentiate the clean up process for clipboard data (e.g. when using tidy as clean up mechanism, copy/paste operation from MS Word 2000 has to be handled differently).

Example:

```
import com.realobjects.eop.applet.custom.CleanUpProcessWrapper;
...
public class CleanUp
    implements CleanUpProcessWrapper
{
    ...
    /**
     * <p>Cleans up the given raw data and returns the cleaned data if no exception
     * occurs.</p>
     * <p>This method should be overridden for a custom implementation of the (X)HTML clean
     * up process.</p>
     * @param strRawData Raw data to be cleaned up
     * @return clean data
     * @throws Exception Thrown if clean up process fails
     */
    public String cleanUp( String strRawData ) throws Exception
    {
        //TODO: implement your clean-up process
    }
    /**
     * <p>Cleans the clipboard by given data raw and returns the cleaned data if no
     * exception occurs.</p>
     * <p>This method should be overridden for a custom implementation of the (X)HTML clean
     * up process.</p>
     * @param strRawData Raw data to be cleaned up
     * @return clean data
     * @throws Exception Thrown if clean up process fails
     */
    public String cleanUpClipBoard(String strRawData) throws Exception
```

```

    {
        //TODO: implement your clean-up process
    }
}

```

- b. Set both the “isclientside” and the “enabled” attributes of the `cleanupprocess` parameter to “true”. This parameter has to be set in the applet’s configuration file. For details, see Configuration File.

Example:

```

<cleanupprocess class="custom.cleanup" isclientside="true" enabled="true"
tidyconfigfile="http://localhost/eop/tidyconfig.txt" />

```

- c. Make sure that the required libraries used for the client-side clean up process (default: tidy.jar and/or tidy.cab) are included in the parameter

Example :

```

<APPLET code=com.realobjects.eop.applet.EditorApplet.class archive="edit-on-
pro.jar,ssce.jar,tidy.jar">
    ...
    <PARAM NAME="cabbase" VALUE="ssce.cab,edit-on-pro.cab,tidy.cab">
    ...
</APPLET>

```

When Tidy is used as client side clean up process, it is configured as follows (if you use the built-in client side clean up process and don't override the clean up process):

- XHTML mode is true
- SmartIndent (for pretty printing) is true
- ShowWarnings is false
- DocType is set to "loose" i.e. the XHTML Transitional DTD is added

All other Tidy settings are the default values (please see the Tidy documentation for further details: <http://www.w3.org/People/Raggett/tidy/>).

Note: If Tidy is used as client side clean up process, all Tidy settings can be modified in the file specified by the “tidyconfigfile” attribute of the “cleanupprocess” tag. For details about the configuration of Tidy, please see <http://tidy.sourceforge.net/docs/Overview.html>.

2.9.2 The Server Side Clean Up Process

The Server Side clean up process can be enabled in one of the following ways:

- a. Set the “url” attribute in the CLEANUPPROCESS tag, to the URL where the clean up process server-side script (CGI/ASP/PHP/JSP file) is available.
- b. Set the “isclientside” attribute to “false” and the “enabled” attribute to “true”.
- c. The HTML data to clean is in the “htmldata” field of the http request.

Example :

```

<cleanupprocess url="http://MyWebServer/Tidy.jsp" isclientside="false" enabled="true" />

```

A Sample of a JSP File that serves as server-side clean up process:

```

<%@page import="javax.servlet.*" %>
<%@page import="java.net.*" %>
<%@page import="java.io.*" %>
<%@page import="java.util.*" %>
<%@page import="org.w3c.tidy.*" %>
<%

```

```

ByteArrayInputStream strIn;
ByteArrayOutputStream strOut;

BufferedInputStream urlIn;

Tidy tidy = new Tidy();
String content = request.getParameter("HTMLDATA");
PrintWriter res = response.getWriter();

try {
    String strResult = "";
    if ( content != null && !content.equals("") ) {
        byte strByte[] = content.getBytes();
        strIn = new ByteArrayInputStream(strByte);
        strOut = new ByteArrayOutputStream();

        ByteArrayOutputStream strErr = new ByteArrayOutputStream();
        tidy.setErrout(new PrintWriter( strErr, true));
        tidy.setShowWarnings( false );
        tidy.setXHTML( true );
        tidy.setSmartIndent( true );

        tidy.parse(strIn, strOut);
        if ( tidy.getParseErrors() > 0 ) {
            strResult = "<TIDYERRORS>\n" + strErr.toString() + "</TIDYERRORS>";
        }
        else {
            strResult = strOut.toString();
        }
    }
    res.write(strResult);
    res.flush();
}
catch (Exception e) {
    System.out.println( "TidyProcess Error : " + (new Date()).toString() );
    e.printStackTrace();
}
%>

```

2.9.3 Sample cases

Following are some sample cases which how show the clean up process works in edit-on Pro:

- **Sample 1, Clean Up Process is able to correct the document**

Document:

```

<html>
  <head>
    <title></title>
  </head>
  <body>
    <p>This is an <strong>unwellformed document</p>
  </body>
</html>

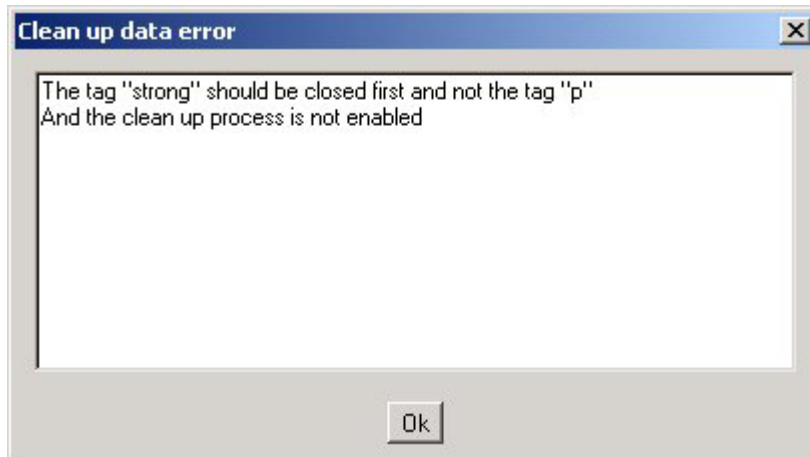
```

Please note that the tag has no closing tag.

If this document is imported to edit-on Pro, the following situations may result:

1. **If clean up process is not enabled**

edit-on Pro will perform internal wellformedness checking and find the unclosed tag. Because the clean up process is not enabled, a message box similar to this one will be displayed and no document will be loaded to edit-on Pro.



2. If the clean up process (either the server side or client side) is enabled

edit-on Pro will perform an internal wellformedness checking to find the unclosed tag, and the document will then be corrected by the clean up process. The wellformed document will then be loaded to edit-on Pro.

The wellformed document resulted from clean up Process will be similar to this:

```
<html>
  <head>
    <meta name="generator" content="HTML Tidy, see www.w3.org" />
    <title></title>
  </head>
  <body>
    <p>This is an <strong>unwellformed document</strong></p>
  </body>
</html>
```

Please note that the closing tag for () is created before </p>

- **Sample 2, Clean Up Process is not able to correct the document**

In the case of Custom XML Tags, these tags must be registered in the clean up process engine, in order to allow the clean up process to correct the document. If the custom XML tags are not registered, the clean up process will fail and return an error message to edit-on Pro. As a result, no document will be loaded. The registration of custom XML tags is only possible when using a server side clean up process.

Document:

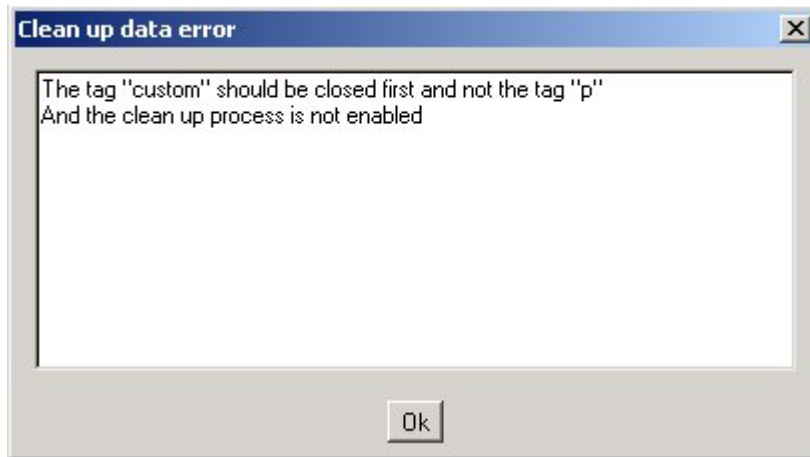
```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <p>This is an <custom>unwellformed document</p>
  </body>
</html>
```

Please note that the <custom> tag has no closing tag.

If this document is imported into edit-on Pro, one of the following cases may occur:

1. If the clean up process is not enabled

edit-on Pro will perform internal wellformedness checking to find the unclosed tag. Because the clean up process is not enabled, a message box similar to this one will be displayed and no document will be loaded into edit-on Pro.



2. If the clean up process (server side) is enabled and the <custom> tag is registered as an inline element.

edit-on Pro will perform an internal wellformedness checking to find the unclosed tag. Because the clean up process is enabled, and the <custom> tag is registered as an inline element, the document will be corrected by the clean up process. The wellformed document will then be loaded into edit-on Pro. Please remember that it is only possible to register the custom XML tag when using a server-side clean up process

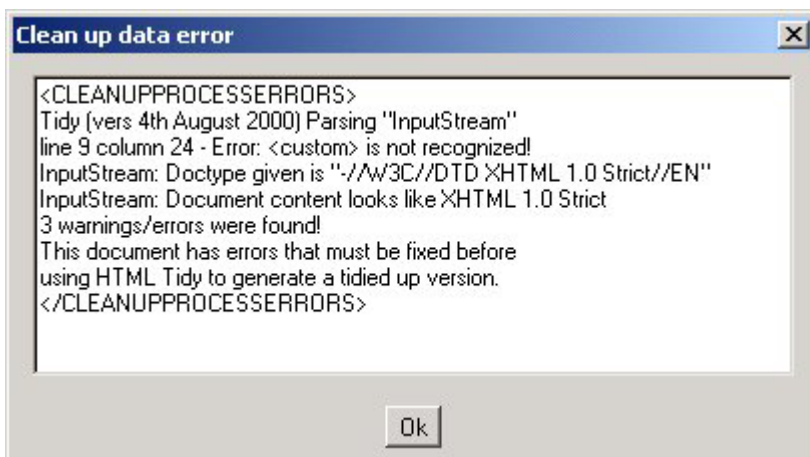
The wellformed document resulting from the clean up Process will be similar to this:

```
<html>
  <head>
    <meta name="generator" content="HTML Tidy, see www.w3.org" />
    <title></title>
  </head>
  <body>
    <p>This is an <custom>unwellformed document</custom></p>
  </body>
</html>
```

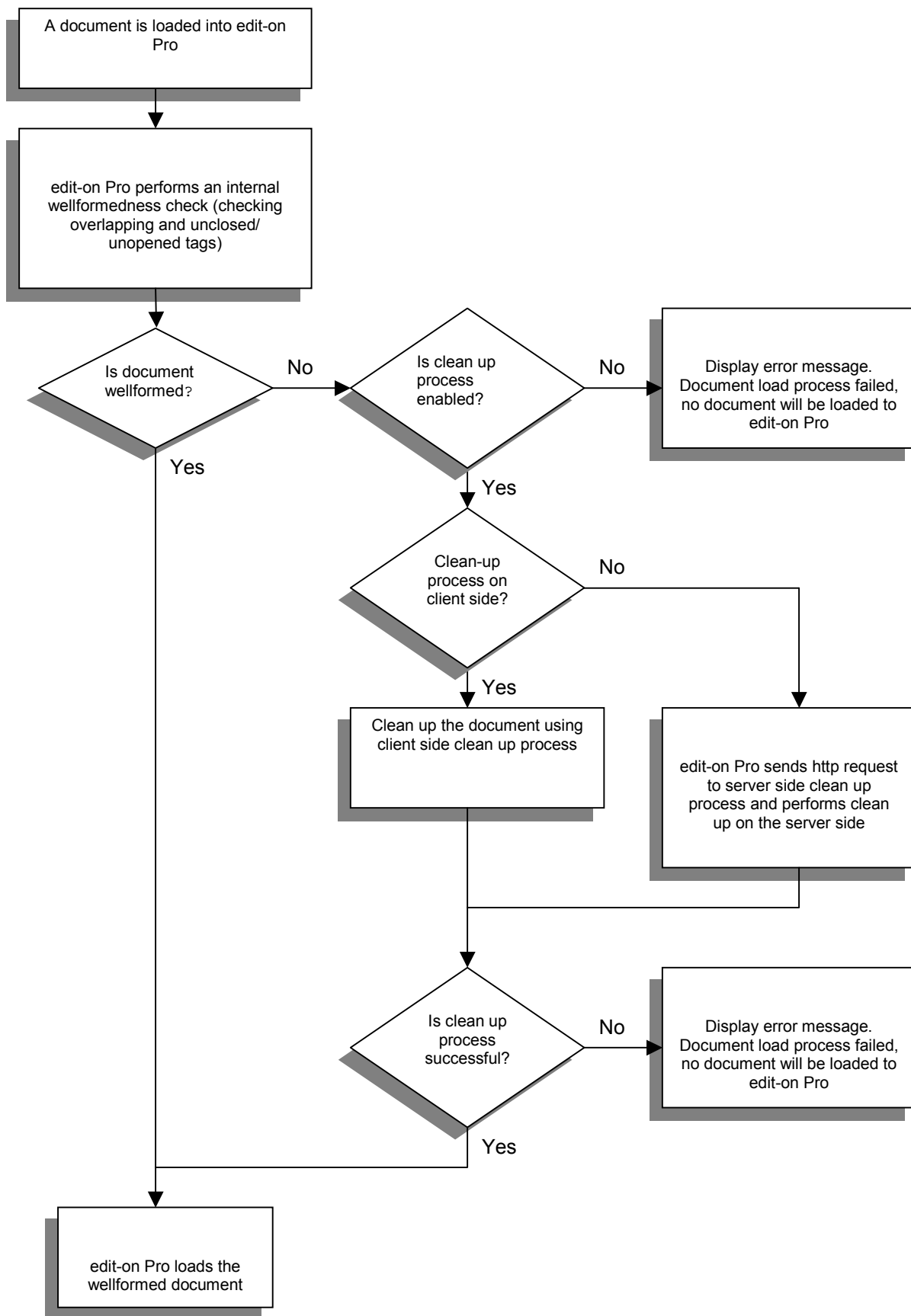
Please note that the closing tag for <custom> (</custom>) is created before </p>

3. If the clean up process is enabled and <custom> tag is not registered.

edit-on Pro will perform an internal wellformedness checking to find the unclosed tags. Because the clean up process is enabled but the <custom> tag is not registered as inline element, the clean up process will not be able to correct the document and it will return an error message. A message box similar to this one will be displayed and no document will be loaded into edit-on Pro.



The following diagram describes the clean up process:



2.9.4 Error Messages

The edit-on Pro integrator should specify any errors encountered in the clean up process inside the `<tidyerrors>` element of the http Response.

2.10 Custom Dialog API

*Note: Implementing Custom Dialogs in Java **requires** knowledge of the Java programming language and general software development knowledge and practice. No technical support can be given for Custom Dialog implementations.*

2.10.1 Introduction

By using the Custom Dialog API of edit-on Pro, it is possible to override certain parts of the standard edit-on Pro user interface. The dialogs that can be overridden are:

- Opening and inserting HTML documents
- Inserting and editing hyperlinks and bookmarks
- Inserting and editing images
- Editing unknown tags (see “Displaying Unknown Tags”)
- Insert Table
- Table properties
- Row properties
- Column properties
- Cell properties
- Color Dialog
- Insert Span Dialog
- Insert Custom Tag Dialog

To facilitate the development of your own custom dialogs, the common base class `CustomDialogWrapper` in the package `com.realobjects.eop.applet.custom` is provided. This class is the base class for all custom dialogs.

To implement a custom dialog, you must perform the following steps:

1. Derive a class from `com.realobjects.eop.applet.custom.CustomDialogWrapper`.
2. Override the abstract method `doDialog()` of the `CustomDialogWrapper` class.
3. Handle the data exchange between edit-on Pro and your class by using `get/setAttribute`.
4. Build your class.
5. Deploy it to the applet server.
6. Specify the dialog you want to override as a PARAM and hand over your classname.

Notes: You must import `com.realobjects.eop.applet.custom.*` in your custom dialog class to build it successfully.

In addition, to build your custom class, `edit-on-pro-devel.jar` must be added to your classpath. It can be found in this folder: `samples\customdialogs\lib` in the unzipped distribution package of edit-on Pro.

Please make sure that you decode data that is retrieved from edit-on Pro using `getAttribute` method before displaying it in your custom dialog. Similarly please make sure that you encode the data from your custom dialog that will be set to edit-on Pro using `setAttribute` method. For more information about helper functions that encode/decode specified string see JavaScript API.

You can use the encode/decode public methods defined in the `EditorApplet` class in your custom dialog. Here's a code snippet showing how to do within a custom dialog:

```
EditorApplet applet = (EditorApplet)getAttribute("applet");
...
String encodedData = applet.encode(DataToEncode);
...
String decodedData = applet.decode(EncodedData);
```

You can find a description of the decode/encode methods in the chapter "JavaScript API".

2.10.2 Reference

`com.realobjects.eop.applet.custom.CustomDialogWrapper` class

Base class:

```
public abstract class CustomDialogWrapper
```

Constructor:

```
public CustomDialogWrapper()
```

Dialog Method

```
public abstract boolean doDialog();
```

This method is called by the applet when the dialog should be invoked. At this time, the attributes described in the table below, which apply for the overridden dialog, may be valid and can be read. However, each value returned should be checked for whether or not it is not NULL.

The boolean return value of this method decides whether or not edit-on Pro will process the results. If the return value is TRUE, edit-on Pro will process the return values. In general, when a "Cancel" button is clicked the `doDialog` method should return false.

Helper functions:

```
protected final void setAttribute( String key, Object value )
public final Object getAttribute( String key )
```

These methods are used to exchange data between edit-on Pro and your custom class. You cannot override them. To remove an attribute from attribute, set use the function `setAttribute(strAttributeName, null)`. The function `getAttribute(strAttributeName)` will return null when the requested attribute does not exist.

All the dialogs in the application that can be customized, must be derived from the class `CustomDialogWrapper`.

The application will create the dialog on demand, and show the dialog by invoking the `doDialog()` method of the `CustomDialogWrapper` derived class. The `doDialog` is where the application expects

the custom dialog to be shown. The dialog should be a modal dialog. On return, the **doDialog** should return a boolean value, to say whether or not the data from the dialog is valid.

The **centerOnComponent**(frame, dialog) is used to specify the central position of the dialog relative to the applet. The frame is the reference to the parent frame, for the dialog to be displayed (see Custom Dialog sample).

Any data needed by the dialogs, will be saved internally in the `CustomDialogWrapper` class, before the application invokes the **doDialog()** methods. The developer can retrieve the data as needed, by calling the **getAttribute**("key") of the `CustomDialogWrapper`. The keys that are valid depend on the dialog type (see below). On return, the dialog should pass back the values that were entered. This can be done using the helper function **setAttribute**("key", value) of the `CustomDialogWrapper`. "key" must always be of the type "String". Note that both the methods **getAttribute**("key") and **setAttribute**("key", value) are case sensitive regarding their arguments.

Dialog Name	PARAM Name	Available Attribute Keys	Value Type	Comment
Open HTML Document	openhtmldlg	frame	Container	read-only!
		applet	Applet	read-only!
		url (required)	String	The URL of the document which should be opened. Should start with http://
Insert HTML Document	inserthtmldlg	frame	Container	read-only!
		applet	Applet	read-only!
		url	String	The URL of the document, which should be inserted at the current cursor location. Should start with http://
Edit Comment Tag	editcommenttagdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		comment	String	The string containing the content of the comment tag.
Edit Tag Properties	edittagpropertiesdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		name	String	read-only!

		attributes	Hashtable	<p>This attribute specifies the attributes of the tag. The hashtable contains:</p> <ul style="list-style-type: none"> ▪ key: specifies the attribute name as String ▪ value: specifies the value of the attribute as String
Insert Hyperlink	inserthrefdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!
		href	String	The A HREF URL where the hyperlink should point.
		target	String	This attribute specifies the named frame for the HREF hyperlink to jump to when activated
		bookmarks	Vector	read-only! This attribute specifies list of bookmark names that exist on current document
You can set other attributes to the dialog in string format using setAttribute function.				
Edit Hyperlink	edithrefdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!

		href	String	The A HREF URL where the hyperlink should point.
		target	String	This attribute specifies the named frame for the HREF hyperlink to jump to when activated
		bookmarks	Vector	read-only! This attribute specifies list of bookmark names that exist on current document
		edit-on Pro will also pass the other available attributes to the dialog in string format. i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application and edit are hyperlink attributes.		
Insert Bookmark	insertbookmarkdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!
		name	String	The bookmark name
		You can set other attributes to the dialog in string format using setAttribute function.		
Edit Bookmark	editbookmarkdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!
		name	String	The bookmark name
		edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application and edit are bookmark attributes.		
Insert Image	insertimagedlg	frame	Container	read-only!
		applet	Applet	read-only!

		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!
		src (required)	String	Indicates the URI to reference the graphic.
		alt (required)	String	This is text to be displayed in place of an image.
		border	String	This controls the thickness of the border around the image (in pixels)
		width	String	This attribute explicitly specifies the width of the graphic in pixels. If set to 0 it will be reset to the original value.
		height	String	This attribute explicitly specifies the height of the graphic in pixels. If set to 0 it will be reset to the original value.
		align	String	This attribute explicitly specifies the alignment of the graphic.
		You can set other attributes to the dialog in string format using setAttribute function.		
Edit Image	editmagedlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		edit	String	read-only!
		src (required)	String	Indicates the URL to reference the graphic.
		alt (required)	String	This is text to be displayed in place of an image.

		border	String	This controls the thickness of the border around the image (in pixels)
		width	String	This attribute explicitly specifies the width of the graphic in pixels. If set to 0 it will be reset to the original value.
		height	String	This attribute explicitly specifies the height of the graphic in pixels. If set to 0 it will be reset to the original value.
		edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application and edit are image attributes.		
Insert Table	inserttabledlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		rowcount (required)	String	This attribute specifies the desired number of rows in the table. The value of this attribute must be greater than zero
		colcount (required)	String	This attribute specifies the desired number of columns in the table. The value of this attribute must be greater than zero


		tableproperties	Hashtable	<p>This attribute specifies the properties of the table. The hashtable contains:</p> <ul style="list-style-type: none">▪ width: specifies the width of the entire table as String (in pixel or percent)▪ border: specifies the width of the table border as String (in pixel)▪ cellpadding: specifies the padding of the cells in the table as String (in pixel)▪ cellspacing: specifies the spacing of the cells in the table as String (in pixel)▪ bgcolor: specifies the background color of the table as String. The String has an RGB format (e.g #808080).▪ bordercolor: specifies the border color of the table as String. The String has an RGB format (e.g #808080).
--	--	-----------------	-----------	--

		cellproperties	Hashtable	<p>This attribute specifies the properties of all cells in the table. The hashtable contains:</p> <ul style="list-style-type: none"> ▪ width: specifies the width of the cells (columns) in the table as String (in pixel or percent) ▪ height: specifies the height of the cells (rows) in the table as String (in pixel or percent) ▪ valign: specifies the vertical alignment of all cells in the table as String
		edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting and application are table attributes.		
Table properties	tablepropertiesdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		width	String	This attribute specifies the desired width of the entire table (in pixel or percent).
		border	String	This attributes specifies the width of the frame around a table (in pixel).
		cellspacing	String	This attributes specifies the spacing between cells (in pixel).
		cellpadding	String	This attribute specifies the amount of space between the border of the cell and its contents (in pixel).

		bgcolor	String	This attributes specifies the background color of the table. The string has an RGB format (e.g #808080).
		bordercolor	String	This attributes specifies the border color of the table. The string has an RGB format (e.g #808080).
		valign	Vector	<p>This attributes specifies the vertical alignment of all cells in the table. The vector contains two elements.</p> <ul style="list-style-type: none"> ▪ The first element indicates the value of the valign attribute as a String. ▪ The second element indicates the homogeneity of the valign attribute as a Boolean. If this element is Boolean.TRUE then all cells in the table have homogen valign attribute values, otherwise it's heterogen. <p>To remove the valign attribute of all cells in the table the dialog must set the first element to null and the second element to Boolean.TRUE.</p> <p>To set the valign attribute of all cells in the table the dialog must set the first element to the new valign value and the second element to Boolean.TRUE.</p>
		edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting and application are table attributes.		
Row properties	rowpropertiesdlg	frame	Container	read-only!
		applet	Applet	read-only!

		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		dialogmode	Integer	read-only!
		customcolor	Vector	read-only!
		height	Vector	This attribute specifies the desired height of the row (in pixel).
		bgcolor	Vector	This attributes specifies the background color of the row. The vector contains two elements. The first element of the vector is a string has an RGB format (e.g #808080).
		valign	Vector	This attributes specifies the vertical alignment of all cells in the row
		<p>All of the vectors in the attributes, except customcolors, contain two elements.</p> <ul style="list-style-type: none"> ▪ The first element indicates the value of the the attribute as a String. ▪ The second element indicates the homogeneity of the attribute as a Boolean. If this element is Boolean.TRUE then all cells in the row have homogen attribute values, otherwise it's heterogen. <p>To remove the attribute of all cells in the row, the dialog must set the first element to null, and the second element to Boolean.TRUE.</p> <p>To set/insert the attribute of all cells in the row, the dialog must set the first element to the new value, and the second element to Boolean.TRUE.</p> <p>edit-on Pro will also pass the other available attributes to the dialog in the above Vector format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application, dialogmode and customcolor are row attributes.</p>		
Column properties	columnpropertiesdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		dialogmode	Integer	read-only!

		customcolor	Vector	read-only!
		width	Vector	This attribute specifies the desired width of the column (in pixel).
		bgcolor	Vector	This attributes specifies the background color of the column. The vector contains two elements. The first element of the vector is a string has an RGB format (e.g #808080).
		valign	Vector	This attributes specifies the vertical alignment of all cells in the column.
		<p>All of the vectors in the attributes, except customcolors, contain two elements.</p> <ul style="list-style-type: none"> ▪ The first element indicates the value of the the attribute as a String. ▪ The second element indicates the homogeneity of the attribute as a Boolean. If this element is Boolean.TRUE then all cells in the column have homogen attribute values, otherwise it's heterogen. <p>To remove the attribute of all cells in the column the dialog must set the first element to null and the second element to Boolean.TRUE.</p> <p>To set/insert the attribute of all cells in the column the dialog must set the first element to the new value and the second element to Boolean.TRUE.</p> <p>edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application, dialogmode and customcolor are column attributes.</p>		
Cell properties	cellpropertiesdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		dialogmode	Integer	read-only!
		customcolor	Vector	read-only!
		height	String	This attribute specifies the desired height of the cell (in pixel).

		width	String	This attribute specifies the desired width of the cell (in pixel).
		bgcolor	String	This attributes specifies the background color of the cell. The String has an RGB format (e.g #808080).
		valign	String	This attributes specifies the vertical alignment of the cell.
		edit-on Pro will also pass the other available attributes to the dialog in string format. i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application, dialogmode and customcolor are column attributes.		
Color Dialog	Colordlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		dialogmode	Integer	read-only!
		customcolors	Vector	read-only!
		standardcolors	Vector	readonly! This attribute specifies the vector of Color to be used as standard colors.
		customcolors	Vector	This attribute specifies the vector of Color to be used as custom colors.
		color	Color	This attributes specifies the selected Color
automaticcolore nabled	Boolean	read only! This attributes specifies whether the automatic color () should be displayed or not.		

		location	Point	readonly! This attributes specifies the location where the color dialog should placed.
		customcoloronly	Boolean	readonly! This attributes specifies whether the custom color should display the standard colors and the custom colros or only the custom colors
		edit-on Pro will also pass the other available attributes to the dialog in string format, i.e. all parameters passed to the dialog except frame, applet, langbase, windowsetting, application, dialogmode and customcolor are column attributes.		
Insert Span	insertspandlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		classes	Vector	read-only!
		class	String	The class name for the span tag
		You can set other attributes to the dialog in string format using setAttribute function.		
Insert Custom Tag	insertcustomtagdlg	frame	Container	read-only!
		applet	Applet	read-only!
		langbase	JwLangBase	read-only!
		windowsetting	WindowSetting	read-only!
		application	Application	read-only!
		customtags	Vector	read-only!
		tagName (required)	String	The tag name for the custom tags

		vAttribute	Vector	This attribute specifies the vector of name-value combination String ("name=value") to be used as the attributes for the custom tags. E.g. if vCustomTag contains "name1=value1" and "name2=value2" then setAttribute("vAttribute", vCustomTag) will result in a new custom tag that has two attributes, namely name1 with value1 as value and name2 with value2 as value.
		You can set other attributes to the dialog in string format using setAttribute function.		

2.10.3 Sample

In this sample, the "Insert Hyperlink" Dialog will be overridden.

This is the custom inserthrefdlg class source:

```
package com.xyz.dialog;

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

// This package must be included, edit-on-pro.jar must be in the class path build the custom
class
import com.realobjects.eop.applet.custom.*;

// Sample custom class implementation

public class InsertHrefDialog extends CustomDialogWrapper
{
    // This method will be called when the user has clicked on the "InsertHyperlink" button

    public boolean doDialog()
    {
        // Get a reference to the parent frame for the dialog
        Container con = (Container) getAttribute( "frame" );

        // Get a reference to the applet,
        // e.g. to get the CODEBASE or to read PARAMS
        Applet apl = (Applet) getAttribute( "applet" );

        PictureDlg dlg = new PictureDlg( (Frame) con, "Href Dialog [Custom]", true );
        /* call setVisible( false ) first and then call centerOnComponent function if
           you would like to center your dialog */
        dlg.setVisible( false );
        centerOnComponent( con, dlg)
        dlg.setVisible( true );

        // Hand back the href attribute
        setAttribute( "href", dlg.list.getSelectedItem() );

        return dlg.retVal;
    }

    protected class PictureDlg extends Dialog
        implements ActionListener
```

```
{
    public PictureDlg( Frame frame, String title, boolean modal )
    {
        super( frame, title, modal );

        setLayout( new BorderLayout() );

        list = new List();
        list.add( "http://www.yahoo.com" );
        list.add( "http://www.google.com" );
        list.add( "http://www.altavista.com" );
        list.add( "http://www.infoseek.com" );
        list.add( "http://www.msn.com" );

        add( list, BorderLayout.CENTER );
        Panel panel = new Panel( new FlowLayout( FlowLayout.CENTER ) );
        Button btn = new Button( "Ok" );
        btn.addActionListener( this );
        panel.add( btn );
        btn = new Button( "Cancel" );
        btn.addActionListener( this );
        panel.add( btn );
        add( panel, BorderLayout.SOUTH );
        setSize( 400, 200 );
    }

    public void actionPerformed((ActionEvent e )
    {
        if ( e.getActionCommand().equalsIgnoreCase( "Ok" ) ) {
            retVal = true;
            dispose();
        } else {
            dispose();
        }
    }

    protected List list = null;
    protected boolean retVal = false;
}
}
```

To activate this custom dialog implementation, build the class file and copy it to the (CODEBASE)/com/xyz/dialog/ directory on the edit-on Pro webserver and add the following PARAM to the applet tag:

```
<PARAM NAME="inserthrefdlg" VALUE="com.xyz.dialog.InsertHrefDialog">
```

3 Integrating edit-on Pro

3.1 License Key Mechanism

edit-on Pro contains an integrated license mechanism based on digital signatures.

The applet will check whether a valid license key is available during run-time.

Together with the edit-on Pro installation package, you will also receive a license key file (*licensekey.xml*). This key file has to be copied into the applet CODEBASE directory, which is the directory where the CAB and JAR files of the applet are placed. It must be named *licensekey.xml*.

The licensekey.xml file must not be changed in any way; it cannot be renamed and the contents may not be edited, or else the key will cease to be valid.

Example:

Assuming the URL of the HTML page where you want to include the applet for the edit-on Pro tag is:

<http://www.yourdomain.com/cms/editpage.htm>

and this page contains the following applet tag:

```
...
<APPLET code="com.realobjects.eop.applet.EditorApplet" archive="edit-on-pro-
signed.jar" height=400 width=750 name=MyEditor>

    <PARAM name="CODEBASE" value="/eopro/">
    <PARAM name="CABBASE" value="edit-on-pro-signed.cab">
.
.
.
</APPLET>
```

then the key file must be named

[licensekey.xml](#)

and must be placed in the directory:

<http://www.yourdomain.com/eopro/>

It must contain a license for the server with the FQDN **[www.yourdomain.com](#)**

Note: A valid key must always be available. However, for development purposes, the applet will also run when used from **<http://localhost>**.

3.2 Calling public edit-on Pro methods with JavaScript

Note: Using the JavaScript API of edit-on Pro, requires that the browser/JavaVM combination supports "LiveConnect". "LiveConnect" refers to JavaScript to Java communication within a web browser. This is supported e.g. by Internet Explorer on Windows, Netscape on Windows and Linux but e.g. not by Internet Explorer on MacOS.

Use of a edit-on Pro in conjunction with a hidden textarea field:

```
<html>
<head>
```

```
<title>JavaScript API/PHP Sample</title>

<link rel="stylesheet" type="text/css" href="../styles/style.css">

<SCRIPT LANGUAGE=javascript>

<!--

    function scriptForm_onsubmit()

    {
        if (document.layers)

            {

                document.HideTXTArea.document.SubmitContent.HTMLText.value =
document.MyEditor.getHTMLData("http://");

                document.HideTXTArea.document.SubmitContent.CSSText.value =
document.MyEditor.getStyleSheet();

                document.HideTXTArea.document.SubmitContent.submit();

            }

        else

            {

                document.SubmitContent.HTMLText.value = document.MyEditor.getHTMLData("http://");

                document.SubmitContent.CSSText.value = document.MyEditor.getStyleSheet();

                document.SubmitContent.submit();

            }

    }

//-----//
//The CSS-Data can not be loaded before HTMLData is completely loaded.
//Thats why "ONEDITORLOADED" and "ONDATALOADED" is used below
//-----//
//This function is called when the applet has finished loading

function loadData()

{

    if (document.layers)

        {

            document.MyEditor.setHTMLData("http://",
document.HideTXTArea.document.SubmitContent.HTMLText.value)

        }

    else
```

```

    {
        document.MyEditor.setHTMLData("http://", document.SubmitContent.HTMLText.value)
    }
}

//This function is called when the editor has finished the loading of HTMLData
function setstyle()
{
    if (document.layers)
    {
document.MyEditor.setStyleSheet (document.HideTXTArea.document.SubmitContent.CSSText.value)
    }
    else
    {
        document.MyEditor.setStyleSheet (document.SubmitContent.CSSText.value)
    }
}

//-->
</SCRIPT>
</head>
<body bgcolor="#FFFFFF" language="Javascript">
<h1>JavaScript API/PHP Sample</h1><br>
<form name="scriptForm" method="post" LANGUAGE="Javascript">
    <applet code="com.realobjects.eop.applet.EditorApplet" id=editor codebase="../../eopro"
name="MyEditor" height="350" width="700" archive="edit-on-pro-signed.jar,tidy.jar,ssce.jar"
VIEWASTEXT MAYSCRIPT>
        <!-- Applet Layout params -->
        <PARAM NAME="WINDOWFACECOLOR" VALUE="#EBF0FF">
        <PARAM NAME="TABPANEACTIVECOLOR" value="#cce3ff">
        <PARAM NAME="WINDOWHIGHLIGHTCOLOR" value="#FFffff">
        <PARAM NAME="LIGHTEDGECOLOR" value="#ebf0ff">
        <PARAM NAME="DARKEDGECOLOR" value="#C5CCFF">
        <PARAM NAME="INNERTEXTCOLOR" value="#000000">
        <PARAM NAME="STARTUPSCREENBACKGROUND" VALUE="#EBF0FF">

```

```

<PARAM NAME="STARTUPSCREENTEXTCOLOR" VALUE="navy">

<!-- End - Applet Layout params -->

<PARAM NAME="cabbase" VALUE="edit-on-pro-signed.cab,tidy.cab,ssce.cab">

<PARAM NAME="locale" VALUE="en_US">

<PARAM NAME="help" VALUE="eophelp/en_US/help_en_US.htm">

<PARAM NAME="bodyonly" VALUE="true">

<PARAM NAME="configurl" VALUE="config.xml">

<PARAM NAME="toolbarurl" VALUE="toolbar.xml">

<PARAM NAME="sourceview" VALUE="true">

<PARAM NAME="sourceviewwordwrap" VALUE="false">

<PARAM NAME="smartindent" VALUE="true">

<PARAM NAME="multipleundoredo" VALUE="true">

<PARAM NAME="oldfontstylemode" VALUE="false">

<PARAM NAME="nbspfill" VALUE="false">

<PARAM NAME="customcolorsenabled" VALUE="true">

<PARAM NAME="tablnbspfill" VALUE="true">

<PARAM NAME="ONEDITORLOADED" VALUE="loadData">

<PARAM NAME="ONDATALOADED" VALUE="setstyle">

</applet>

<br>

<INPUT name="button1" type="button" value="Submit" onClick="scriptForm_onsubmit();">

<br> <br>

<a href="../index.htm" target="_top">Back to Samples Index</a>

<!-- This hidden textarea field will receive the HTMLData on submitting the form. -->

<!-- The layer is used to hide textarea in Netscape 4.x -->

</form>

<layer ID="HideTXTArea" visibility="hide">

<form name="SubmitContent" method="post" action="view.php">

<textarea name="HTMLText" cols="1" rows="1" style="visibility:hidden;"><? echo
stripslashes($GLOBALS['HTTP_POST_VARS']['HTMLText']);?></textarea>

<!--This hidden textarea field will receive the CSSData on submitting the form.-->

<textarea name="CSSText" cols="1" rows="1" style="visibility:hidden;"><? echo
stripslashes($GLOBALS['HTTP_POST_VARS']['CSSText']); ?></textarea>

</form>

</layer>

</body>

```

</html>

Notes:

- This code is taken from the “JavaScript API/PHP Sample which comes with the edit-on Pro samples package.
- As layers are only supported by Netscape 4.x, a distinction must be made between the different browser types.

3.3 Event Handling with JavaScript

Note: Using the Event Handling of edit-on Pro with JavaScript requires that the browser/JavaVM combination supports "LiveConnect". "LiveConnect" refers to JavaScript to Java communication in a web browser. This is supported e.g. by Internet Explorer on Windows, Netscape on Windows and Linux but e.g. not by Internet Explorer on MacOS.

edit-on Pro provides event handlers which will execute JavaScript functions when fired.

1. ONEDITORLOADED

This event handler will be fired after edit-on Pro finishes its loading event.

2. ONDATALOADED

This event handler will be fired after the setHTMLData JavaScript API function of edit-on Pro finishes its data loading event. See “JavaScript API”.

3. ONDATAPOSTED

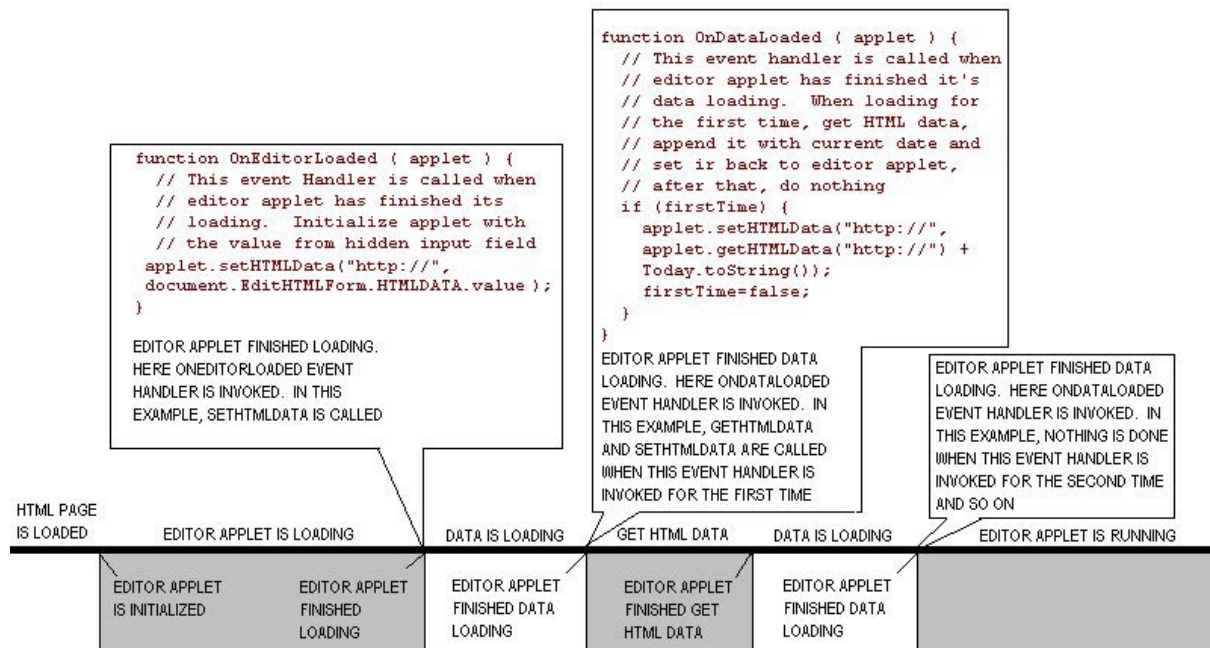
This event handler will be fired after the HTML data has been successfully sent from the editor, to an URL address specified in the parameter GETHTMLDATAURL.

4. ONDATAPOSTEDERROR

This event handler will be fired after the HTML data has not been sent successfully, from the editor to an URL address specified in the parameter GETHTMLDATAURL.

All event handlers will execute a JavaScript function, specified in the applet parameter. See “Parameter API”. These event handlers function like public edit-on Pro methods, called in the “onLoad” attribute of HTML body tag; therefore, the calling of public edit-on Pro methods in the “onLoad” attribute of the HTML body tag is no longer necessary.

Following is a detailed illustration of the event handling process. This illustrates sample code from the chapter “Calling public edit-on Pro methods by JavaScript”. For the complete sample code, see “Calling public edit-on Pro methods by JavaScript”.



Sample Code for the JavaScript function “OnEditorLoaded”, using its parameter:

```

<HTML>

<HEAD>
<TITLE>edit-on Pro Multi Instances</TITLE>

<SCRIPT language="JavaScript">

  //If you use an Eventhandler like "ONEDITORLOADED", make sure that the "MAYSCRIPT"
  //statement is in the <applet> tag:

  //The following function is called by the "OnEditorLoaded" - applet parameter of the
  //first applet

  //The applet parameter of this function contains a reference to the applet which
  //has called it (in this case a reference to the first one "myEditor1")
  function HeadLineLoad( applet )
  {
    //Within the next statement, you can also use the name of the applet instead of
    //the reference "applet"

    //We make the editor load a string via "setHTMLData" function of the applet:

    applet.setHTMLData( "http://", "<p>Please enter your headline here... </p>" );
  }

  //The following function is called by the "OnEditorLoaded" - applet parameter of the
  //second applet
    
```

```

//The applet parameter of this function contains a reference to the applet which
//has called it (in this case a reference to the second one "myEditor2")

function ArticleLoad( anotherapplet )
{
//The name of the function paramater does not have to be "applet" each time.....

//We make the applet load some text out of a hidden text field, which is in the form
//below :

    anotherapplet.setHTMLData( "http://", document.myForm.HTMLDATA.value );
}
</SCRIPT>
</HEAD>
<BODY bgcolor="#CCCCCC">
<h1>JavaScript Demo (PHP)</h1><br>
<table>
<tr valign="center">
<td><h4>Headline</h4></td>
<td>
<APPLET code="com.realobjects.eop.applet.EditorApplet" codeBase="../../eopro/"
id="myEditor1" name="myEditor1" archive="edit-on-pro-signed.jar,ssce.jar" width="600"
height="160" VIEWASTEXT MAYSCRIPT>

<!-- The MAYSCRIPT statement is very important here. Without it the applet doesn't call
any event handler. -->

<PARAM NAME="cabbase" VALUE="edit-on-pro-signed.cab,ssce.cab">

<PARAM NAME="locale" VALUE="en_US">

    .. .. .
    .. .. .

<PARAM NAME="oneditorloaded" value="HeadLineLoad">

</APPLET>

</td>
</tr>
<tr>
.. .. .
.. .. .

<APPLET code="com.realobjects.eop.applet.EditorApplet" codeBase="../../eopro/"
id="myEditor2" name="myEditor2" archive="edit-on-pro-signed.jar,ssce.jar" width="600"
height="160" VIEWASTEXT MAYSCRIPT>

```

```
<!-- The MAYSCRIPT statement is very important here. Without it the applet doesn't call
any event handler. -->
```

```
<PARAM NAME="cabbage" VALUE="edit-on-pro-signed.cab,ssce.cab">
```

```
<PARAM NAME="locale" VALUE="en_US">
```

```
.. . . . .
```

```
.. . . . .
```

```
<PARAM NAME="oneditorloaded" value="ArticleLoad">
```

```
</APPLET>
```

```
</td>
```

```
</tr>
```

```
... ..
```

(For complete code see the “Multiarticle” Sample)

Note:

- To enable Event Handling with JavaScript, the `MAYSCRIPT` applet attribute must be declared inside the applet tag.
- JavaScript functions specified in the applet parameter contain a function parameter. This parameter includes a reference to the applet, which has called the JavaScript function. You can use the parameter within that function instead of the name of the applet.

3.4 Using direct HTTP connections

If the JavaScript API cannot be used, for example, because the client platform doesn't support LiveConnect, there is an additional way to interface with the edit-on Pro applet. There are two applet parameters (see “Applet Parameter API”), `SETHTMLDATAURL` and `GETHTMLDATAURL`, which enable this function. If `SETHTMLDATAURL` is defined, edit-on Pro will perform a request to the specified URL at initialization time, and then load the data from there into its HTML buffer.

The opposite way (saving back the data to a server), can be realized by setting `GETHTMLDATAURL` to point to a custom CGI script. Additionally, the `GETHTMLDATAURL` has to be enabled in the toolbar definition file. The user is then able to click on the save button in the toolbar, and edit-on Pro will perform a HTTP POST request to the URL where `GETHTMLDATAURL` points to. The POST request will be application/x-www-form-urlencoded, and will contain the content of edit-on Pro's HTML buffer in a form variable called `HTMLDATA`.

The server application must reply `<result>OK</result>` back to edit-on Pro.

Sample `GETHTMLDATAURL` handler (ASP)

```
<%@ language="JavaScript" %>
<%
    var filePath = Server.MapPath("save.htm");
    var serverObj = Server.CreateObject("Scripting.FileSystemObject");
    var writeObj = serverObj.OpenTextFile( filePath, 2, true );

    writeObj.Write( Request.Form("HTMLDATA") );
    writeObj.Close();
%>
<result>OK</result>
```

Sample `GETHTMLDATAURL` handler (PHP3)

```
<?php
    $myfile = fopen("save.htm", "wb");

    fwrite($myfile, stripslashes($HTMLDATA));
    fclose($myfile);
?>
<result>OK</result>
```

Sample GETHTMLDATAURL handler (JSP)

```
<%@page import="javax.servlet.*" %>
<%@page import="java.net.*" %>
<%@page import="java.io.*" %>
<%@page import="java.util.*" %>
<%
    try {
        /* enter the target file */
        FileWriter fw = new FileWriter( application.getRealPath("save.htm"));
        String strContent = request.getParameter("HTMLDATA");
        OutputStreamWriter outWriter = new OutputStreamWriter( outStream,
                                                                request.getCharacterEncoding() );

        outWriter.write(strContent);
        outWriter.flush();
        outWriter.close();
        fw.flush();
        fw.close();
        PrintWriter res = response.getWriter();
        res.write("<result>OK</result>");
        res.flush();
        System.out.println("Request content Length : " + nLength + " Content Length : " +
                           nRecLength );
    }
    catch (Exception e) {
        System.out.println( "SavePost Error : " + (new Date()).toString() );
        e.printStackTrace();
    }
%>
```

3.5 Image Root

In general, images that are used within a particular web-based application or website, are stored on the same web server. An exception to this is, for example, an ASP (application service provider) which offers image hosting services, allowing its customers to store images for online auctions.

If you work with images that are located on the home server, usually relative image paths are used. The `IMAGEROOT` Applet parameter defines the root from which all relative image paths will be expanded.

Let's assume we have following scenario. We run a simple CMS to maintain the Website. edit-on Pro is used as the CMS editing tool. Among others there are the following directories on the Web server (<http://www.realobjects.com>):

- **/doc/examples** Some HTML files which are used within our Website reside here. One file is called `example.htm`. The content of the file consists of some text and a photo (=../pictures/image.jpg).
- **/pictures** Besides various other JPGs you will find the file `image.jpg` here.
- **/screenshot** Besides various other JPGs you will find another file `image.jpg` here.
- **/applet/eop** Here you will find the file `editor.htm` which includes the edit-on Pro `<APPLET>` tag. <http://www.realobjects.com/applet/eop> is the document base because the document base is the path where the HTML file that contains the `<APPLET>` tag resides.

Within the `<APPLET>` tag we set the parameter "imageroot" to:

```
<PARAM NAME="IMAGEROOT" VALUE="http://www.realobjects.com/pictures/">
```

Because the parameter "imageroot" is defined, the following will apply:

- **Import Document**

The *imageroot* parameter is used to find all images which use a relative path as “src” attribute.

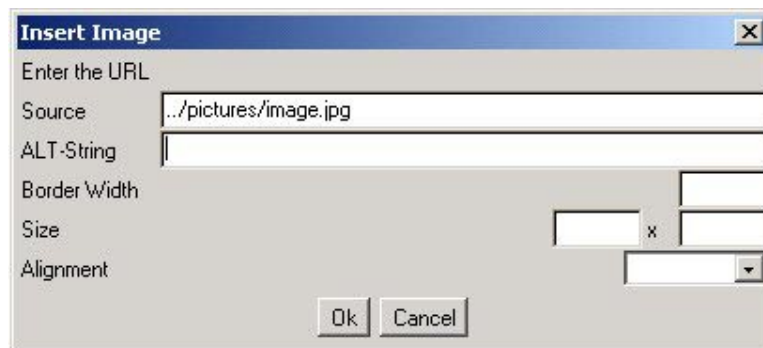
Example: We assume that you import the document
<http://www.realobjects.com/doc/examples/example.htm> into the editor

If the imported document contains the `` tag, edit-on Pro will load the image from <http://www.realobjects.com/pictures/image.jpg>

If the imported document contains the `` tag, edit-on Pro will use the complete URL address to find the image, in this case <http://www.realobjects.com/screenshot/image.jpg>

- **Insert Image**

The *imageroot* parameter is used to find all images with the source entry as a relative path.



Example:

If the insert picture icon at the toolbar is clicked, and the source entry is `../pictures/image.jpg`, as seen above, edit-on Pro will assume that the image is located at <http://www.realobjects.com/pictures/image.jpg>.



If the insert picture icon at the toolbar is clicked, and the source entry is `http://www.realobjects.com/screenshot/image.jpg`, as seen above, edit-on Pro will use the complete URL address to find the image, in this case <http://www.realobjects.com/screenshot/image.jpg>.

- **Export Action** through either
 - The toolbar save icon
 - The getHTMLData function
 - A change from WYSIWYG to HTML view

The image addresses contained in the document will be exported unchanged.

Example: If in both of the examples above, the user exports the document, then the results will be exactly as shown above.

```


```

These default settings are used if the imageroot parameter is not specified.

- **Open document**

The hostname of the document is used to find all images with a relative path.

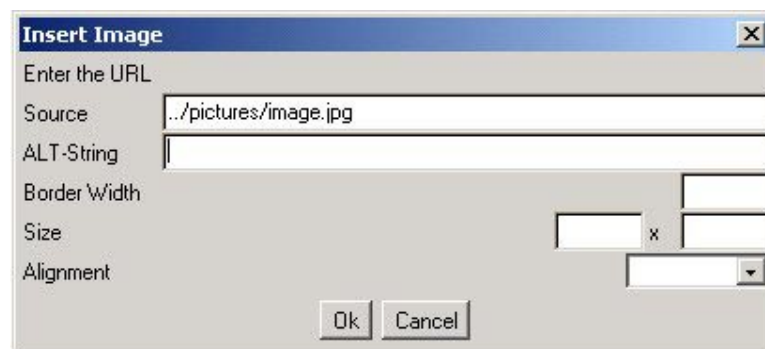
Example: We assume that the user imports the document
<http://www.realobjects.com/doc/examples/example.htm>

If the imported document contains the `` tag, edit-on Pro will assume that the image is located at
<http://www.realobjects.com/doc/pictures/image.jpg>.

If the imported document contains the `` tag, edit-on Pro will use the complete URL address to find the image, in this case
<http://www.realobjects.com/screenshot/image.jpg>.

- **Insert Image**

The hostname of the applet document base is used to find all images with a relative path.



Example:

If the insert picture icon at the toolbar is clicked and the source entry is `../pictures/image.jpg`, as seen above, edit-on Pro will assume that the image is located at <http://www.realobjects.com/applet/pictures/image.jpg>, because the document base is <http://www.realobjects.com/applet/eop>.



If the insert picture icon at the toolbar is clicked and the source entry is `http://www.realobjects.com/screenshot/image.jpg`, as seen above, edit-on Pro will use the complete URL address to find the image, in this case `http://www.realobjects.com/screenshot/image.jpg`.

- **Export**

- From WYSIWYG mode to HTML view mode
If the editor applet contains an active document, the URL address will be truncated relative to the current document's URL if possible. A document is active if the user imports a document.

Example: We assume the user imports the document
`http://www.realobjects.com/doc/examples/example.htm`.

Using the import document example above, the result will be

```


Using the insert example above, the result will be


```

But if no document is active, the URL address will be complete. No document is active if the user clicks the new icon in the toolbar.

Example: We assume the user creates a new document by clicking the new icon.

Using the import document example above, the result will be

```


```

Using the insert picture example above, the result will be

```


```

- Through the save button, the `jsp/cgi/asp/php` address is specified in the `GETHTMLDATAURL` parameter of the applet.
- WYSIWYG mode is active

The image address will always be a complete URL address because we use `getHTMLData("http://")`.

Example: Using the import document example above, the result will be

```


Using the insert picture example above, the result will be


```

- HTML View mode is active

The data in the HTML view text area will be transferred to the server unchanged.

- `getHTMLData` (reference url address) through java script
- WYSIWYG mode is active

The Image address will be relative to the reference address used in `getHTMLData` if possible

Example: We assume the user imports the document by calling the function `setHTMLData` (“<http://www.realobjects.com/doc/examples/example.htm>”).

Using the import document example above, the result will be

```


```

Using the insert picture example above, the result will be

```


```

- HTML View mode is active

The data in the HTML view text area will be transferred to the server unchanged.

3.6 Applet Caching Mechanism

Applet Caching is the mechanism used to cache a specific Java Applet permanently on the local disk. Once the applet has been cached, it no longer needs to be downloaded every time it is referenced again. This mechanism will decrease the loading time. When a new version of the applet is available, the new applet will be downloaded again automatically, thus providing version update capability. Following are the steps to follow to deploy this mechanism. The version strings have to be updated manually, whenever a new edit-on Pro version is deployed. Please consult the `readme.txt` for details about the exact version number and strings.

3.6.1 Applet Caching Mechanism using the Microsoft Java Virtual Machine (Java Package Manager)

To download a signed applet into a local directory on your computer and to force the JVM to search in that specific directory first (before deciding to download the applet from the web site), a slight change to the HTML codes that reference the applet should be made. The Java Package Manager technology is used to download digitally signed applets in the form of CAB files into a specific directory on your local drive.

There are four applet parameters: *namespace*, *useslibrary*, *useslibrarycodebase*, and *useslibraryversion*. These parameters should be implemented in order to instruct the JVM to check for the installed distribution unit of the applet. Should it fail to find it there, then you should download the applet from the website. Here is an example showing a way to implement this:

```
...
<APPLET code=com.realobjects.eop.applet.EditorApplet.class codeBase="." id=editor name=editor
width="750" height="450" VIEWASTEXT>
  <PARAM NAME="cabbage" VALUE="edit-on-pro-signed.cab">
  <PARAM NAME="namespace" VALUE="roEOP3">
  <PARAM NAME="useslibrary" VALUE="RealObjects edit-on Pro 3.1">
  <PARAM NAME="useslibrarycodebase" VALUE="edit-on-pro-signed.cab">
  <PARAM NAME="useslibraryversion" VALUE="3,1,240,0">
  ...
</APPLET>
...
```

- The *namespace* parameter has to specify an unique identifier for the applet. The namespace you specify is used by the Java Package Manager to create a unique application namespace layer above the namespace of packages and classes. With an application namespace, there is no concern about class name collisions with other software developers.
- If you wish to use the Package Manager, you need include the *useslibrary* attribute in the applet tag. This parameter specifies the name of the Java Package that is downloaded to your local directory.
- The *useslibrarycodebase* parameter specifies the cabinet file where the Java classes reside. This must be a digitally signed CAB file.

- The `useslibraryversion` parameter specifies the version of the applet file. This parameter will be analyzed, and then conduct an update should a newer version become available. You have to manually update this version number in your code, when you install a new version of edit-on Pro on your server.

IMPORTANT: A cabinet file indicated both by the `useslibrarycodebase` and the `cabbage` parameter will be loaded twice, as it will be loaded once from the cache and then from the server. If you want to load it from cache only, make sure it is only included in the `useslibrarycodebase` parameter, and not in the `cabbage` attribute.

A security warning dialog box will be displayed when the signed applet is referenced. For more information, please refer to Java Security Considerations. The downloaded signed applet will be stored in the `<WINDOWS DIRECTORY>\downloaded program files`, in the form of a Java package. Users may delete the Java package in this directory should a resident local version of the applet no longer be needed. The browser will check this directory to determine whether a download is required or not. If in a later version, the signed applet becomes available, this Java package will be updated and the new version downloaded again. If there is no newer version of the signed applet available, then the page that references this applet will use this Java package instead of downloading it every time.

3.6.2 Applet Caching Mechanism using the Sun JRE (Java Plug In Caching Mechanism)

To download an applet into a local directory onto your computer, and to force the JVM to search in it first before deciding to download the same applet from the web site, a slight change to the HTML code that references the applet should be made.

There are three applet parameters: `cache_archive`, `cache_version`, and `cache_option`. These parameters should be implemented to direct the JRE to check for the installed distribution unit of the applet. Should it fail to find it there, then just download the applet from the website. Here is an example of how this works:

```
...
<APPLET code=com.realobjects.eop.applet.EditorApplet.class codeBase="." archive="" id=editor
name=editor width="750" height="450" VIEWASTEXT>
  <PARAM NAME="cache_archive_ex" value="edit-on-pro-signed.jar;3.1.240.0,
  tidy.jar;3.1.240.0, ssce.jar;3.1.240.0">
  <PARAM NAME="cache_option" VALUE="plugin">
  ...
</APPLET>
...
```

The `cache_archive_ex` parameter specifies the cabinet file where the Java classes reside. The value of `cache_archive_ex` has the following format:

```
cache_archive_ex =
"<jar_file_name>;<preload(optional)>;<jar_file_version>,<jar_file_
name>;<preload(optional)>;<jar_file_version>,..."
```

The `preload` and `jar_file_version` attributes can be set in any order after the `jar_file_name`. They must be separated by the delimiter ";"

Class files and resources will be searched in the following order from the `.jar` files specified by the HTML parameters

1. `cache_archive_ex`
2. `cache_archive`
3. `archive`

The `cache_option` parameter specifies the cache to be used to store the class files and resources.

The “cache_option” attribute can have one of three values:

No

Disable applet installation. Always download the file from the web server.

Browser

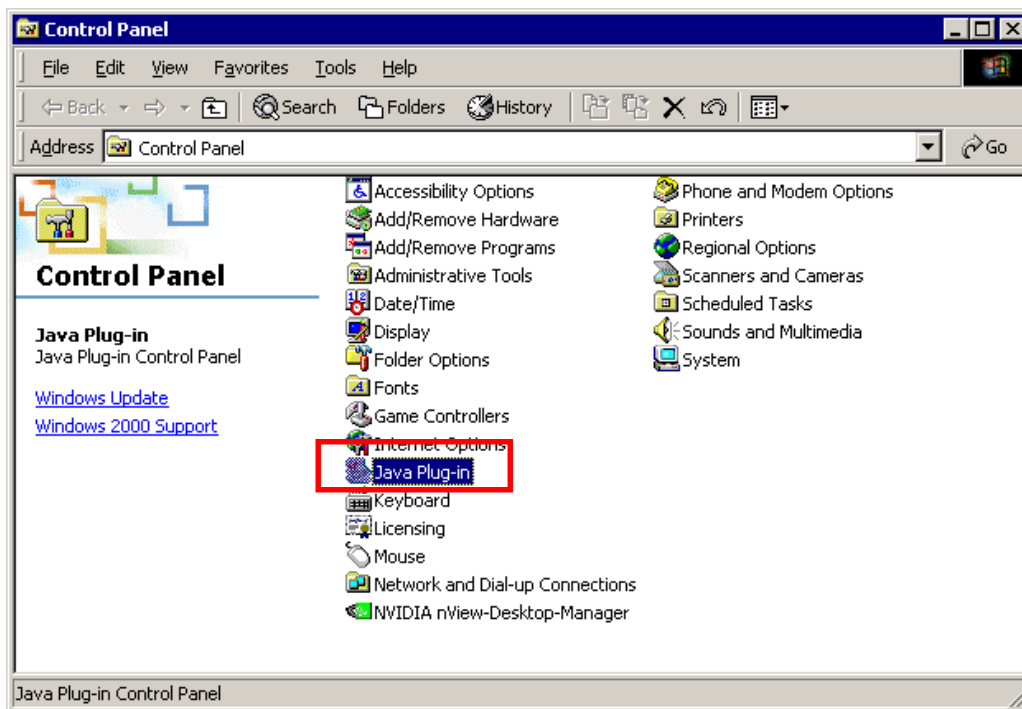
Run applets from the browser cache (default).

Plugin

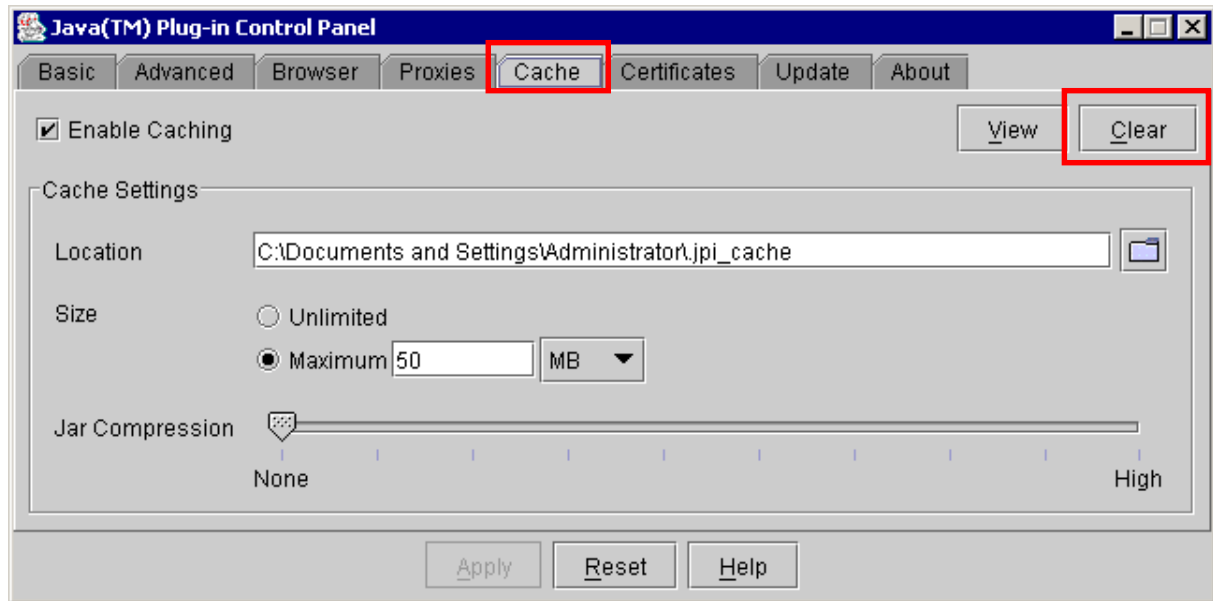
Run applets from the new Java Plug-in cache.

When the cache_option parameter is set to the value of plugin, the downloaded and signed applet will be stored in the <DOCUMENT AND SETTINGS DIRECTORY>\<USER NAME>\java_plugin_AppletStore\<JAVA PLUG IN VERSION>\jar, for example: C:\Documents and Settings\john.d\java_plugin_AppletStore\1.4.1\jar. Should a resident local version of the applet no longer be needed, the user may delete the cached Java file in this directory using the Java Plug In control from control panel.

To delete any cached jar files from the local directory, go to the Windows control panel, and locate the Java Plug In control icon. Click on the icon, and a dialog box will be displayed.



In the dialog box, click on the “Cache” tab. Then click on the “Clear JAR Cache” button to clear the content of the caching directory.



The browser will check the caching directory to determine whether or not a download is required. If later on, a newer version of the signed applet becomes available, a new JAR file will be downloaded. If no newer version of the signed applet is available, then the page that references this applet will use this jar file, instead of downloading it every time. Please note that the new JAR file will not override the existing JAR file, as they are saved using different file names. To avoid the accumulation of JAR files over several versions of applet, just clear the JAR cache as explained earlier.

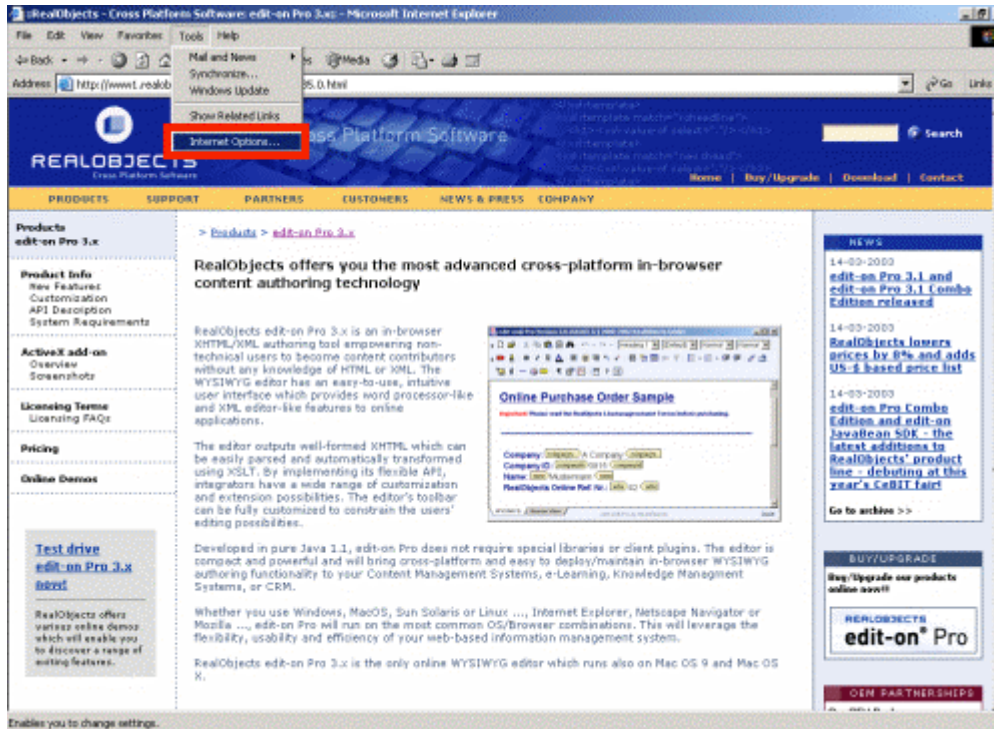
4 Using edit-on Pro

4.1 Browser Settings

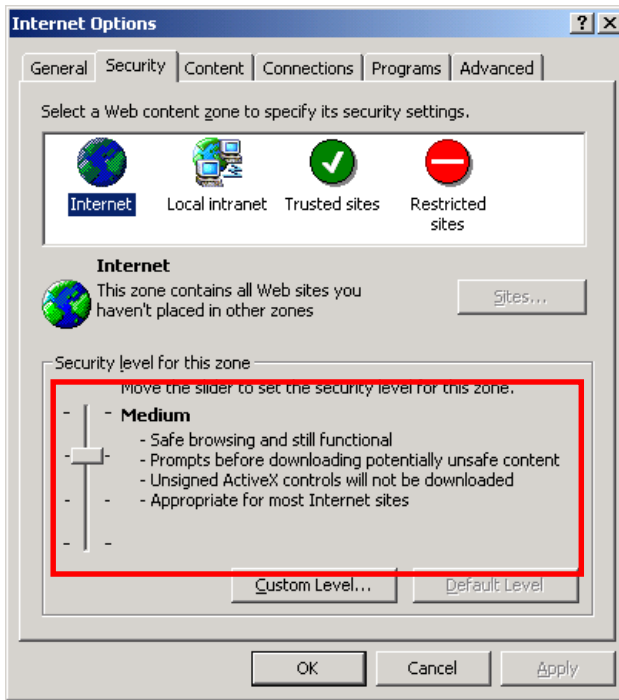
In order to use edit-on Pro, your browser must be set-up to support Java. The standard settings of most browsers are OK for edit-on Pro. However, following you will find a guide to properly set up your browser.

4.1.1 Setting Up Microsoft Internet Explorer with Microsoft Java VM

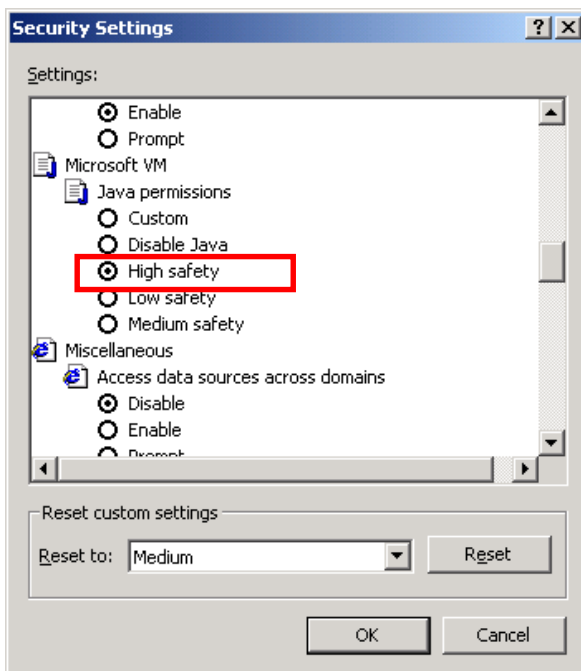
1. Go to “Tools”, “Internet Options ...”



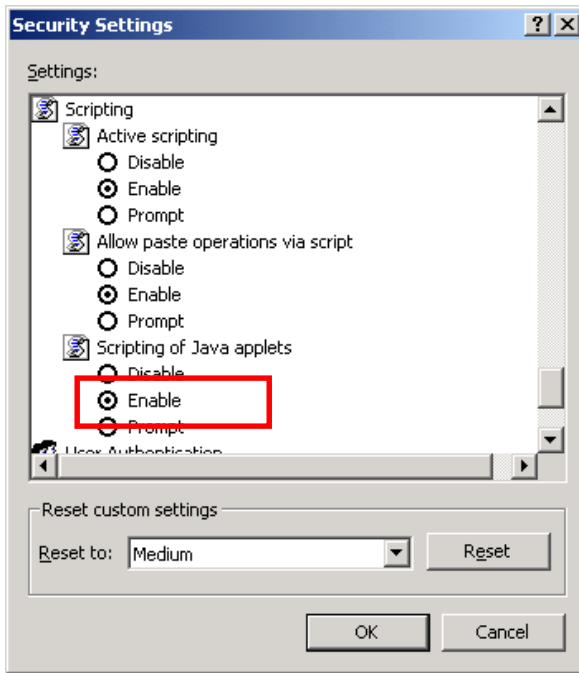
2. In the “Internet Options” dialog, go to the “Security” tab. The security level for the content zone from where you load edit-on Pro must at least be set to "Medium" (default setting for Internet zone). edit-on Pro will not work in "High" security level.



3. If you have to use "Custom settings" for certain reasons, make sure that "Microsoft VM" – "Java Permissions" is **not** set to "Disabled Java". The default setting for "Medium Level" security is "High safety" for "Java Permissions" – edit-on Pro runs with this setting.



4. Also make sure that the "Scripting" – "Scripting of Java Applets" setting is enabled

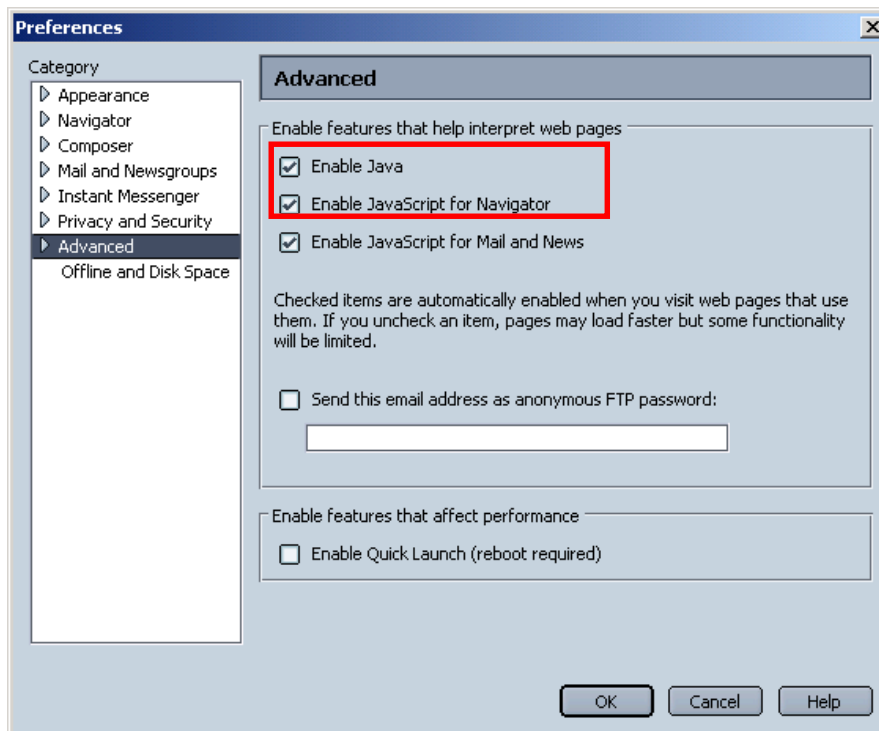


4.1.2 Setting Up Netscape 6.x, 7.x & Mozilla 1.x with Sun JRE 1.4.1

1. Go to “Edit”, “Preferences ...”



2. In “Preferences” go to the “Advanced” category. Make sure that both “Enable Java” and “Enable JavaScript for Navigator” are checked.



4.2 Using the edit-on Pro Editor

4.2.1 Hot Keys

The following hot keys are available within the editor:

Hot Key	Functionality
Ctrl + A	Select All
Ctrl + End	Go to the end of document
Ctrl + Home	Go to the beginning of document
Ctrl + R	Refresh the edit-on Pro editor
Ctrl + X	Copy selected text into clipboard and delete it from editor (cut)
Ctrl + C	Copy selected text into clipboard
Ctrl + V	Paste clipboard content into editor
Ctrl + Arrow Keys	Go to next/previous word
Insert	Toggle between insert and overwrite mode
Shift + Enter	Insert a tag (available only in WYSIWYG view)

4.2.2 Using The Copy/Cut/Paste functionalities

The Copy/Cut/Paste function inside the WYSIWYG View, work internally with preserving styles/formatting etc.


Formatted text can be copied/cut/pasted from/to Microsoft Word and Microsoft Excel only if the Sun JRE 1.4.1 or better is used on Windows. To allow the user to Copy/Cut/Paste formatted text, the "htmlformatclipboard" parameter must be set in the "config.xml" configuration file. See "Configuration File".

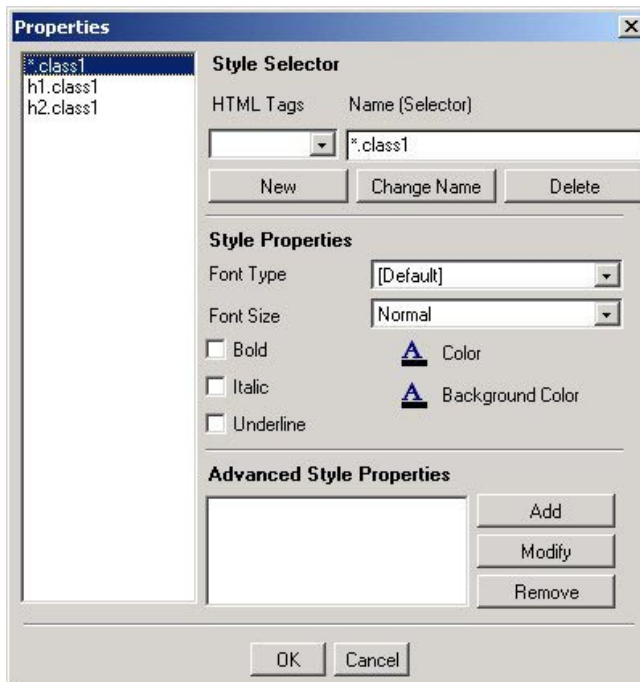
4.2.3 Style Sheets Support

4.2.3.1 Style Properties Dialog

The style sheet can be edited using the style properties dialog. To display the icon which allows access to the style properties dialog in the toolbar, the appropriate definitions must be put in the toolbar.xml file. Following is an example of a corresponding fragment of the toolbar.xml file.

```
<toolbar>
...
  <icon paramname="PROPERTIESTDIALOG" />
...
</toolbar>
```

Parameter:	PROPERTIESTDIALOG
Syntax:	PROPERTIESTDIALOG;icon_URL
Default Icon:	
Default Icon URL:	icons/properties.gif
Description:	When TRUE, users may bring up a settings dialog, where they can adjust the style sheet properties, such as adding new styles, deleting existing styles, renaming styles and changing style attributes.



Dialog Screenshot:

The Style Properties dialog provides an interface which allows the user to create, change and delete styles. Please note that the styles listed in this dialog are the editable styles. For more information about editable and read only styles, see "Editable Styles and Read Only Styles".

- **Adding new styles**

The styles can be added using two types of selectors:

- HTML Tags

To add a new style to the document using the HTML tags selector, simply select the HTML Tags selector from the HTML Tags choice menu and click 'New'. Please note that as the user selects the HTML Tag, the 'Name (Selector)' text field will display the corresponding selector. The new style will have the corresponding HTML Tag as its selector, and it will be listed in the list on the left side of the dialog.

To create a new style the user should specify the style properties of the style. See "Modifying styles".

e.g.:

Suppose the user selects 'strong' from the HTML Tags choice menu, and then clicks 'New', the new style will have 'strong' as its selector.

Note: 'strong' defines a new style that can only be applied to the "strong" HTML Tag

- Name (Selector)

To add a new style to the document using the name selector, simply type the selector in the 'Name (Selector)' text field and click 'New'. Please note that when something is typed in the 'Name (Selector)' text field, the HTML Tag choice menu will be emptied. The new style will have the corresponding name selector, and will be listed in the list on the left side of the dialog.

To create a new style, the user should specify the properties of the style he/she wants to create. To learn more about how to create properties of the style, see "Modifying styles".

For example,

Suppose the user types *.class1 from the HTML Tags choice menu, and then clicks 'New'. The new style will have *.class1 as its selector.

Note: *.class1 defines a new style that can be applied to any HTML Tag that has the 'class=class1' attribute.

Note: Adding a new style will not take effect, if none of the style properties are specified. Please make sure that the user specifies at least one property, for the new style to take effect.

• Modifying styles

Styles can be modified by altering its properties. The style properties dialog allows two types of style modifications:

- Style Properties

Style properties allow the user to modify simple properties of the style. Styles which may be modified in the Style properties area are:

- Font Type
- Font Size
- Bold
- Italic
- Underline

- Font Color
- Background Color (currently edit-on Pro only supports background color in table, td, and body tags)
- **Advanced Style Properties**

Advanced style properties, are properties which are not present within the style selector. The variation of properties which can be modified is virtually limitless, but edit-on Pro supports only a fragment of the available style sheet definitions. For a list of supported styles, see “Styles Supported by edit-on Pro”.

The user can add name-value pair attributes, using the ‘Add’ button in the ‘Advanced Style Properties’. Pressing this button will open a new dialog which requests the name-value pair of the style’s attribute. Simply typing in the text field and then clicking OK, will insert the name-value pair into the ‘Advanced Style Properties’ list.

The user can also modify and remove the existing advanced style properties. To modify existing advanced style properties, simply select the style from the advanced style properties list and then click ‘Modify’. A new dialog will be opened; the user can make the appropriate modifications and then press OK to apply them. To remove existing advanced style properties, select the style from the advanced style properties list and click ‘Remove’.

Note: Not all advanced style properties are supported by edit-on Pro. For a list of supported styles, see “Styles Supported by edit-on Pro”.

- **Changing a style’s name**

To change a style’s name:

- Select the style from the list on the left side of the screen
- Type the new name in the ‘Name (Selector)’ text field or select an item from HTML Tags choice menu
- Click ‘Change Name’

- **Deleting a style**

To delete a style’s name:

- Select the style from the list on the left side of the screen
- Click ‘Delete’

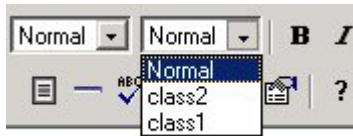
4.2.3.2 Style Sheet Class Selector Choice Menu

The style sheet can be applied to the HTML tag in the current cursor position using the style sheet class selector choice menu. To display the choice menu in the toolbar, the appropriate definitions must be put in the toolbar.xml file. Following is an example of a fragment of the toolbar.xml file.

```
<toolbar>
...
<choice paramname="STYLESHEET" />
...
</toolbar>
```

Parameter: **STYLESHEET**
Syntax: **STYLESHEET**
Description: Allows the user to select stylesheet classes, which can then be applied into a selection of text using a drop down list box. This feature is disabled when there is no selection.

Screen Shot of a toolbar portion displaying the style sheet:



For more information see “Toolbar Definition”.

The Style Sheet Class Selector Choice Menu is context sensitive, meaning that the classes listed in the choice menu will change in relation to the HTML tags at the cursor position.

e.g:

These styles are specified in the document:

```
<style>
    *.class1 { ... }
    h2.class2 { ... }
    h1.class3 { ... }
</style>
```

When the cursor is placed in the document under any HTML tags, the choice menu will list:

- Normal
- class1

When the cursor is placed in the document under <h1> tags, the choice menu will list:

- Normal
- class1
- class3

When the cursor is placed in the document under <h2> tags, the choice menu will list:

- Normal
- class1
- class2


Please note that the choice menu will be disabled when there is a selection. To apply style to a selection, use Insert Span.

4.2.3.3 Insert Span

The style sheet can be applied to the HTML tag in the current cursor position, when there is a selection using the ‘Insert Span’ button. To display the insert span icon in the toolbar, the appropriate definitions must be put in the toolbar.xml file. Following is an example of a fragment of the toolbar.xml file.

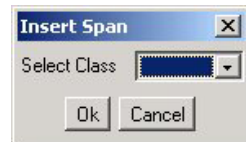
```
<toolbar>
    ...
    <icon paramname="INSERTSPAN" />
    ...
</toolbar>
```

Parameter: **INSERTSPAN**
 Syntax: INSERTSPAN;Icon_URL

Default Icon: 

Default Icon URL: icons/insertspan.gif


Description: When clicked, a dialog box will ask for the stylesheet class to apply to the span. The available classes will be generic and span context classes. After pressing OK, a span tag pair will be inserted.



Dialog Screenshot:

The user may use this feature, to insert a span tag around a selection of text, and to apply a style sheet class into that tag. Please note that the classes listed in the class choice menu of the Insert Span dialog are the styles which have generic selectors (e.g.: *.class1) and/or the styles that have span selectors (e.g.: span.class1)

4.2.4 Displaying Unknown Tags

Unknown tags will be preserved and displayed as yellow marks (e.g. unknown unknown) within the WYSIWYG View, if the  toolbar button is activated. An “unknown tag property” dialog will be displayed when clicking the right mouse button on the yellow marks. The contents of the unknown tag can be edited while changes will be preserved.

4.2.5 Online Help

To make the online help accessible from the help button on the toolbar, please make sure that the following settings are done:

- The HELP API parameter is set to specify the URL of the help file in HTML format. The help will be displayed in a new browser window when the user clicks on the HELP icon. See “Parameter API”.
- The HELP icon must be activated by setting the HELP parameter in the toolbar definition file. See “Toolbar Definition”.

The Online Help can be found in the “eophelp” sub-directory of the edit-on Pro main installation directory (usually “eopro”). It is available in US English, Spanish, French and German. Please feel free to customize the online help to your needs.

4.3 Java Security Considerations

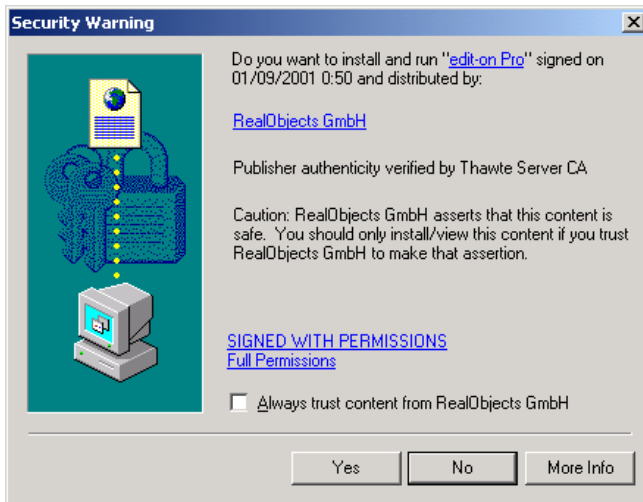
Java applets run in a sandbox mode. By default, applets have no access to system resources outside the directory/environment from which they were launched; however, a signed applet can access local system resources, as allowed by the local system’s security policy. The Java Development Kit 1.2 provides security tools so that end users and system administrators can sign applets and applications. Thus, they may defining their local security policy, by specifying in a policy file how much access to local system resources a signed applet or application can have.

4.3.1 Running edit-on Pro as a signed applet

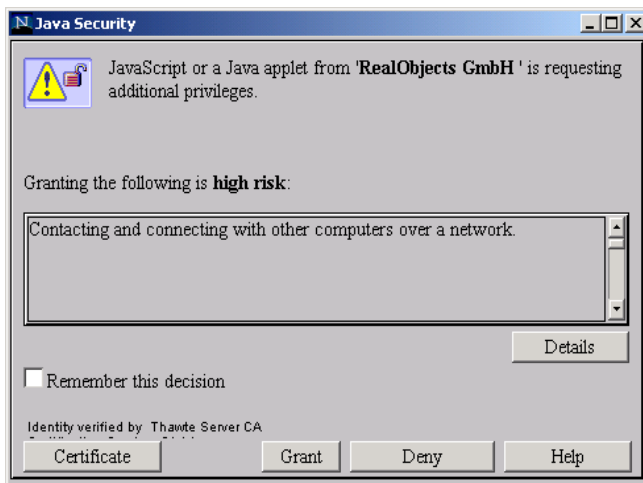
The edit-on Pro applet archive has been signed with a Thawte digital certificate (a Verisign company), to bypass the Java applet sandbox. Signing an applet archive guarantees that the applet is from the certificate owner (RealObjects in this case), and that the archive is the original and has not been tampered with. A signed applet will have access to system resources, which enables Cut and Paste feature to the system clipboard. To run a signed edit-on Pro applet:

1. Run edit-on Pro using the signed applet archive (edit-on-pro-signed.cab / edit-on-pro-signed.jar).
2. A “Security Warning” notification dialog box in Microsoft Internet Explorer or “Java Security” dialog box in Netscape Navigator 4.7 or Netscape Navigator 6.x will appear, similar to the following:

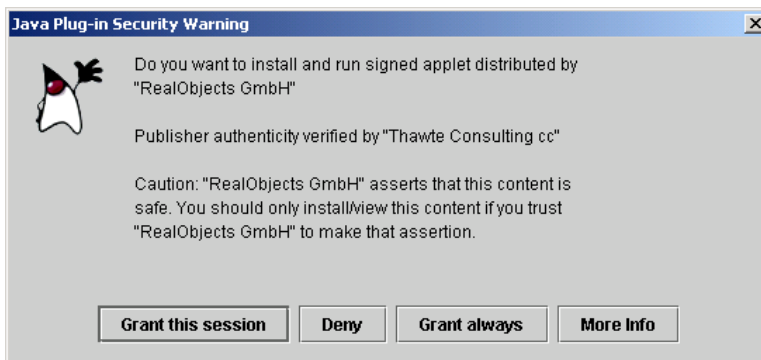
“Security Warning” dialog box in Microsoft Internet Explorer 5.0



“Java Security” dialog box in Netscape Navigator 4.7



“Java Security” dialog box in Netscape Navigator 6.x



3. In Microsoft Internet Explorer 5.0:

To continue loading the edit-on Pro signed applet, click “Yes”. To abort loading, click “No”. The “Always trust content from RealObjects GmbH” option, should be checked if you want your browser to insert RealObjects in as a trusted publisher, therefore preventing it from displaying this dialog box again.

In Netscape Navigator 4.7:

To continue loading the edit-on Pro signed applet, click “Grant”. To abort loading, click “Deny”. The “Remember this decision” option, should be checked if you want your browser to put RealObjects in as a granted vendor, therefore preventing it from displaying this dialog box again.

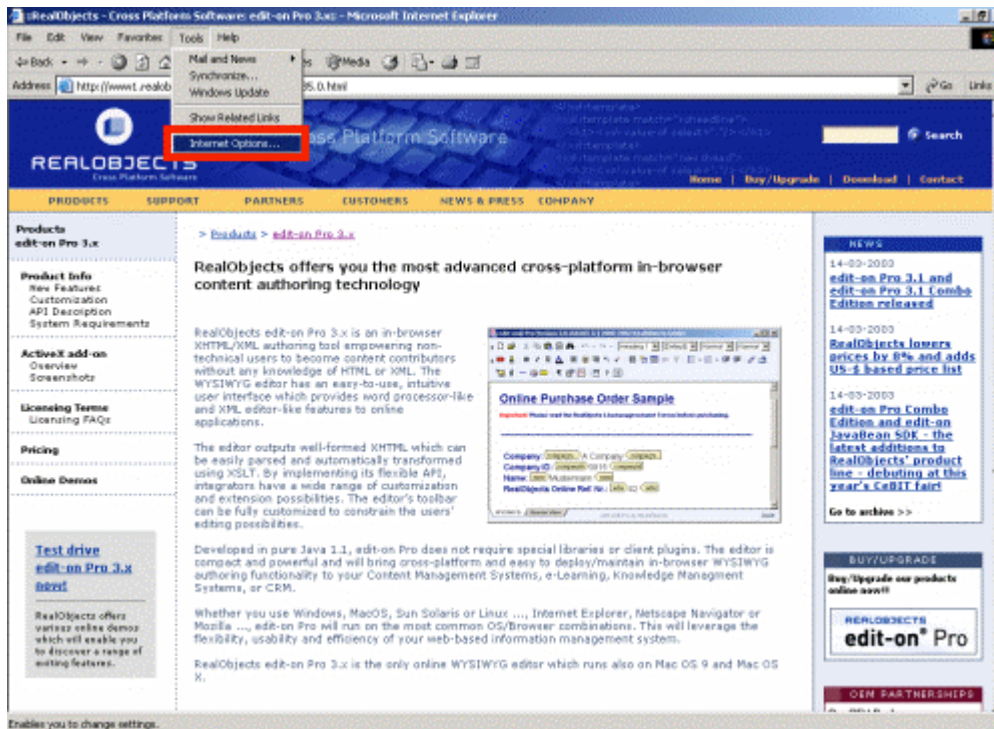
In Netscape Navigator 6.x:

To continue loading the edit-on Pro signed applet, click “Grant this session”. To abort loading, click “Deny”. The “Grant always” button, should be clicked if you want your browser to put RealObjects in as a granted vendor, thus preventing it from displaying this dialog box again.

4.3.2 Removing RealObjects from Trusted Publisher in Microsoft Internet Explorer

If RealObjects is registered as a trusted publisher for your browser, the “Security Warning” notification dialog box will not appear to warn you for the edit-on Pro signed applet. To remove RealObjects from your browser’s list of trusted publishers and display this dialog box again, follow the following steps:

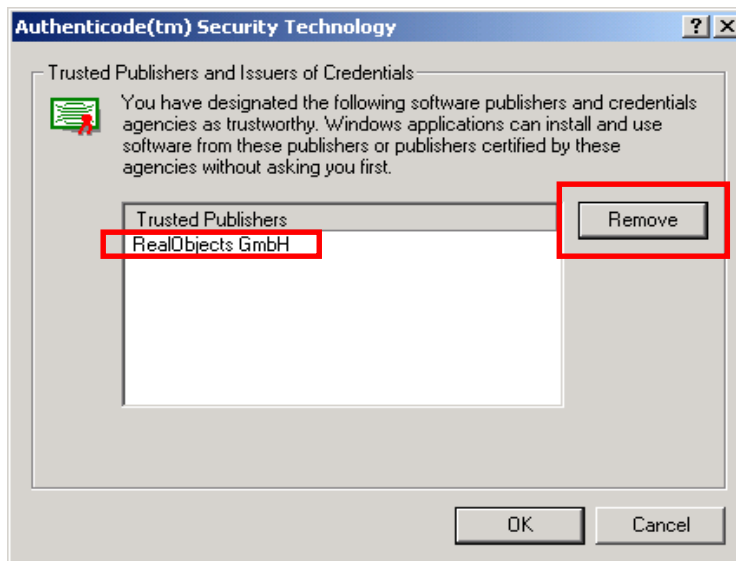
1. Go to “Tools”, “Internet Options ...”



2. In the “Internet Options” dialog, go to “Content” tab. Click the “Publishers...” button in the “Certificates” frame to list the trusted publishers in your browser.



3. In the “Authenticode™ Security Technology” dialog box, select “RealObjects GmbH” from the “Trusted Publisher” list. Click “Remove” to remove it from the trusted publisher list for your browser. The “Security Warning” notification dialog box should appear every time you load edit-on Pro signed applet until you again register RealObjects as trusted publisher.

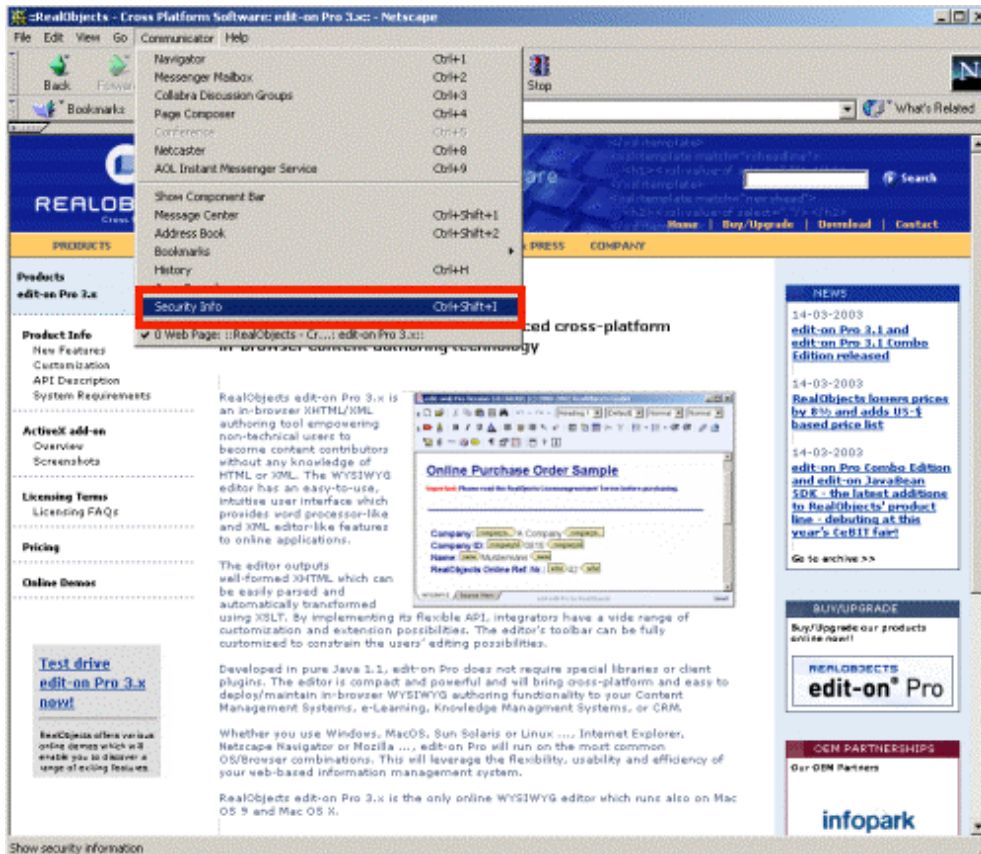


4.3.3 Removing RealObjects from Granted Vendors in Netscape Navigator

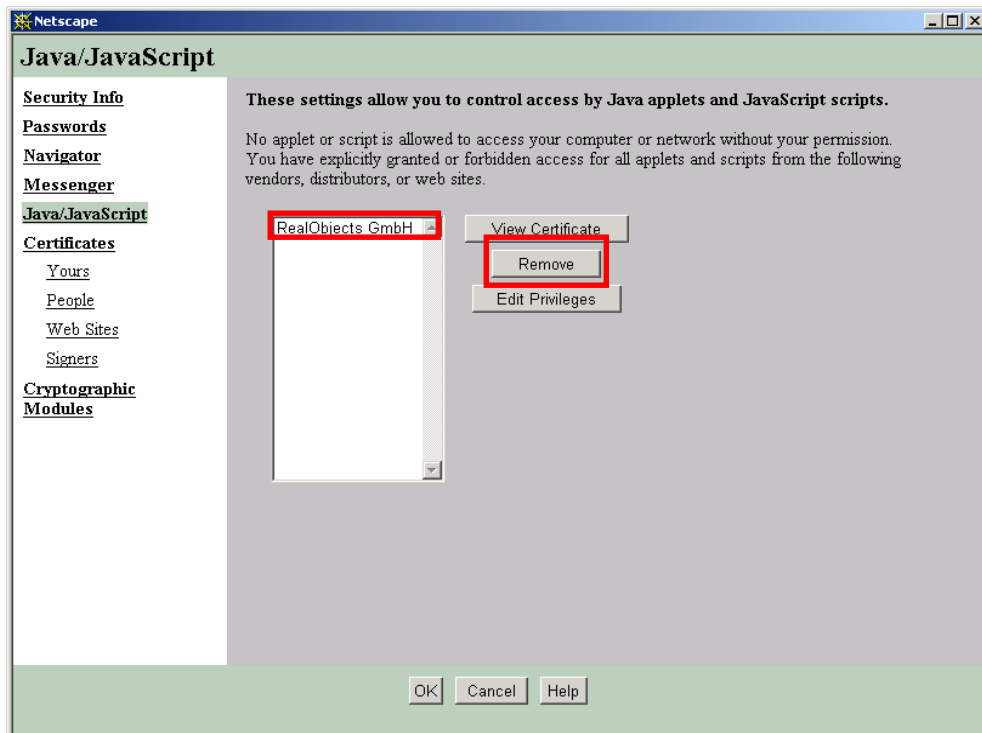
If RealObjects is registered as a granted vendor for your browser, the “Java Security” notification dialog box will not appear to warn you for the edit-on Pro signed applet. To remove RealObjects from your browser’s list of granted vendors and display this dialog box again, follow the following steps:

4.3.3.1 Removing RealObjects from Granted Vendors in Netscape Navigator 4.x

1. Go to “Communicator”, “Security Info”

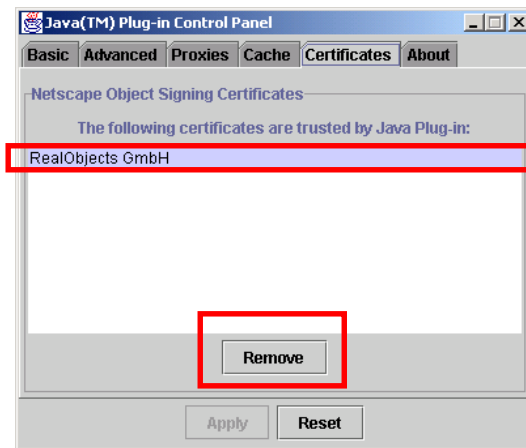


- In the “Security Info” dialog, select “Java/JavaScript”. Select “RealObjects GmbH” from the list of vendors that are granted access, and then click “Remove”. The “Java Security” notification dialog box should appear every time you load the edit-on Pro signed applet, until you add RealObjects to the trusted publishers again.



4.3.3.2 Removing RealObjects from Granted Vendors in Netscape Navigator 6.1

In Netscape Navigator 6.1, Granted Vendors can be removed using the “Java Plug In 1.3.1” Control Panel in “Control Panel”. Click on the “Certificates” tab and select “RealObjects GmbH” from the list of granted vendors. Then click “Remove” to remove it. The “Java Security” notification dialog box should appear every time you load the edit-on Pro signed applet, until you choose to grant access RealObjects again.



Removing RealObjects from Granted Vendors in Netscape 6.1

4.4 Enabling the Spelling Checker

To activate the spelling checker in edit-on Pro, please complete the following steps :

1. Add the spelling checker library to the HTML Applet tag

```
<APPLET code="com.realobjects.eop.applet.EditorApplet.class" archive="edit-on-pro-
signed.jar,ssce.jar" name=MyEditor>
  <PARAM name="CODEBASE" value="http://www.yourdomain.com/eopro/">
  <PARAM name="CABBASE" value="edit-on-pro-signed.cab,ssce.cab">
  .
  .
  .
</APPLET>
```

2. Add the SPELLINGCHECKERS element and SPELLINGCHECKER element(s) into the configuration file (config.xml). This will specify the languages supported by the spelling checker, and the respective URL address of the spelling checker properties files in relation to the applet code base. The spelling checker properties files define which options to set and which dictionaries/lexicons to open. You can specify several languages for the spelling checker to support. Change the parameter value to change the dictionary/lexicon.

```
<spellingcheckers>
  <spellingchecker name="english" properties="english.properties" default="true"/>
  <spellingchecker name="german" properties="german.properties" default="false"/>
</spellingcheckers>
```

3. Add the parameter SPELLCHECK in the button ordering file, in order to make the spellchecking icon visible.
4. Copy the dictionary/lexicon files specified in the spelling checker properties file to place it in its proper location within the web server.

4.5 Spelling Checker Properties

A spelling checker properties file might contain the following keys:

Lexicon :

- MainLexicon
- UserLexicon

Options :

- CASE_SENSITIVE_OPT
- IGNORE_ALL_CAPS_WORD_OPT
- IGNORE_CAPPED_WORD_OPT
- IGNORE_MIXED_CASE_OPT
- IGNORE_MIXED_DIGITS_OPT
- IGNORE_NON_ALPHA_WORD_OPT
- REPORT_DOUBLED_WORD_OPT
- REPORT_MIXED_CASE_OPT
- REPORT_MIXED_DIGITS_OPT
- REPORT_UNCAPPED_OPT
- SPLIT_CONTRACTED_WORDS_OPT
- SPLIT_HYPHENATED_WORDS_OPT
- SPLIT_WORDS_OPT
- STRIP_POSSESSIVES_OPT
- SUGGEST_SPLIT_WORDS_OPT
- IGNORE_DOMAIN_NAMES_OPT
- ALLOW_ACCENTED_CAPS_OPT

4.6 MainLexicon and UserLexicon

The spelling checker opens main lexicons using keys named MainLexicon n , where n is a number ranging from 1 to 99. The first main lexicon must be specified by MainLexicon1, and any other main lexicons must be numbered sequentially starting with MainLexicon2. It will be stopped when the MainLexicon sequence numbers break.

UserLexicon and MainLexicon property settings have the following form:

```
{Main|User}Lexicon $n$ =name [, access method] [, format]
```

n: Sequence number of the main or user lexicon, ranging from 1 to 99. Lexicons are opened in the specified sequence.

name: Name of the lexicon file, resource, or URL (depending on the *access method*).

access method: Method used to access the named lexicon:

- file: The lexicon is opened as a local disk file (default).

- resource: The lexicon is opened as a resource using java.lang.Class's getResourceAsStream method.
- url: The lexicon is opened as a URL using java.net.URL's openStream method.

format: The format of the lexicon. The format defaults to "t" unless the extension part of the *name* parameter is "clx" or "uclx", in which case the default format is "c":

t: text lexicon

c: compressed lexicon.

The spelling checker accesses the lexicon files relative to the code base. The lexicon files can reside in the same directory as the applet, or in a sub-directory.

Currently available MainLexicons:

- American English MainLexicon
- British English MainLexicon
- Canadian English MainLexicon
- German MainLexicon
- French MainLexicon
- Spanish MainLexicon
- Finnish MainLexicon
- Dutch MainLexicon
- Swedish MainLexicon
- Italian MainLexicon
- Norwegian
- Danish
- Brazilian
- American Medical
- British Medical

4.6.1 Creating a Custom Lexicon (Dictionary)

A custom Lexicon can be created by making a new text lexicon file, or by altering an old one. The text lexicon is a mere text file which can be edited using any text file editor, e.g. Notepad. Custom words can be added by simply appending the word at the end of the word list, inside the text lexicon file. The "ADDSPELLCHECKWORDURL" tag in the configuration file, can be used to specify the CGI/ASP/PHP/JSP URL, which appends the new word to the related text lexicon file. See "Configuration File".

4.7 Spelling Checker Options

The option value can be either "true" or "false." The option is set if the value is "true." The option's default value is used if the corresponding key is not included in the Properties.

The options affect the way the spelling checker operates. In most cases, options are enabled by setting their values to true, and disabled by setting their values to false.

- **ALLOW_ACCENTED_CAPS_OPT**: If set to true, capital letters containing accents (e.g., *Être*) are considered acceptable. If set to false, words containing accented capitals are considered misspelled. Should be set to false when checking French Canadian text using Wintertree Software's French dictionary, and set to true in all other cases. Setting this option to false will degrade performance. Default: true.
- **IGNORE_CAPPED_WORD_OPT**: Set to true if words should be ignored (skipped) if they begin with an upper-case letter. Set to false if the words should be checked for spelling errors. Example: If set to true, ignore *Clarkson*; if set to false, check *Clarkson*. Default: false.
- **IGNORE_ALL_CAPS_WORD_OPT**: Set to true if words consisting entirely of upper-case letters should be ignored (skipped). Set to false if the words should be checked for spelling errors. Example: If set to true, ignore *ASAP*; if set to false, check *ASAP*. Default: false.
- **IGNORE_DOMAIN_NAMES_OPT**: Set to true if words that appear to be Internet domain names should be ignored (skipped). Set to false if the words should be checked for spelling errors. Words are considered to be Internet domain names if they contain at least one "dot" (".") and at least two alpha-numeric characters in a row. Example: If set to true, ignore *wintertree-software.com*; if set to false, check *wintertree-software.com*. Default: false.
- **IGNORE_MIXED_CASE_OPT**: Set to true if words containing an unusual mixture of upper- and lower-case letters should be ignored (skipped). Set to false if such words should be checked for spelling errors. Example: If Set to true, ignore *PrintScreen*; if set to false, check *PrintScreen*. Default: false.
- **IGNORE_MIXED_DIGITS_OPT**: Set to true if words containing a mixture of letters and digits should be ignored (skipped). Set to false if such words should be checked for spelling errors. Example: If set to true, ignore *Win95*; if set to false, check *Win95*. Default: false.
- **IGNORE_NON_ALPHA_WORD_OPT**: Set to true if words that contain no alphabetic characters should be ignored (skipped). Set to false if the words should be checked for spelling errors. Example: If set to true, ignore *12345*; if set to false, check *12345*. Default: true.
- **REPORT_UNCAPPED_OPT**: Set to true if uncapitalized words which only exist in capitalized form in the lexicon, should be reported (via `UNCAPPED_WORD_RSLT`). Set to false if uncapitalized words should not be reported. Example: If set to true, report *canada*; if set to false, do not report *canada*. Default: true.
- **REPORT_MIXED_CASE_OPT**: Set to true if words containing an unusual combination of upper- and lower-case letters should be reported (via `MIXED_CASE_WORD_RSLT`). Set to false if such words should not be reported. Example: If set to true, report *TUesday*; if set to false, do not report *TUesday*. Default: false.
- **REPORT_MIXED_DIGITS_OPT**: Set to true if words containing a combination of letters and digits should be reported (via `MIXED_DIGITS_WORD_RSLT`). Set to false if such words should not be reported. Example: If set to true, report *June5*; if set to false, do not report *June5*. Default: false.
- **REPORT_DOUBLED_WORD_OPT**: Set to true if two occurrences of the same word in a row should be reported (via `DOUBLED_WORD_RSLT`). Set to false if doubled words should not be reported. Example: If set to true, report *the the*; if set to false, do not report *the the*. Default: false.
- **CASE_SENSITIVE_OPT**: Set to true if words with different letter-case patterns should be treated as different words. Set to false if words containing different case patterns should be treated as identical. Setting this option to false will degrade performance. Example: If set to

true, treat *Canada* and *canada* as two different words; if set to false, treat *Canada* and *canada* as the same word. Default: true.

- **SPLIT_HYPHENATED_WORDS_OPT**: Set to true if hyphens ("-") should be treated as word separators, and thus each sub-word checked separately. This word splitting is done only if the hyphenated form of the word does not exist in any open lexicon. The word is considered correctly spelled if all sub-words are correctly spelled. Set to false if hyphenated words should be checked in their entirety. Example: If set to true, and *bright-blue* was not found in the open lexicons, check both *bright* and *blue* and treat *bright-blue* as correctly spelled if both words are found; if set to false, report *bright-blue* as misspelled if not found in the open lexicons. Default: true.
- **SPLIT_CONTRACTED_WORDS_OPT**: Set to true if apostrophes should be treated as word separators, and thus each sub-word checked individually. This word splitting is done only if the contracted form of the word does not exist in any open lexicon. The word is correctly spelled if all sub-words are correctly spelled. This option is intended for use with languages which allow ad hoc contractions (e.g., French and Italian). Set to false if contracted words should be checked in their entirety. Example: If set to true, and *quell'anno* was not found in the open lexicons, check both *quell* and *anno*, and treat *quell'anno* as correctly spelled if both words are found; if set to false, report *quell'anno* if not found in the open lexicons. Default: false.
- **SPLIT_WORDS_OPT**: Set to true if words should be treated as a series of concatenated sub-words, and thus each sub-word checked individually. This word splitting is done only if the original word is not found in any open lexicon. The word is correctly spelled if all sub-words containing two or more characters are correctly spelled. This option is intended for use with languages which allow ad hoc concatenation of words (e.g., German and Finnish). Set to false if words should be checked in their entirety. Example: If set to true, and *dumptruckdriver* was not found in the open lexicons, attempt to locate valid sub-words (in this case *dump*, *truck*, and *driver*), and treat *dumptruckdriver* as correctly spelled if all sub-words are found; if set to false, report *dumptruckdriver* if it is not found in the open lexicons. Default: false.
- **STRIP_POSSESSIVES_OPT**: Set to true if possessives of the form 's and s' should be removed from words before checking their spelling. The main lexicons included with the Sentry SDKs contain no possessive word forms, so this option should be enabled when using these lexicons. Set to false if words should be checked with their possessives intact. Default: true.

SUGGEST_SPLIT_WORDS_OPT: Set to true if the suggest method should attempt to split words into two valid sub-words. Set to false if split words should not be suggested. Example: If set to true, suggest *the boy* as a replacement for *theboy*; if set to false, do not suggest *the boy*. Default: false.

4.8 Example Properties File

```
#Sentry Spelling Checker Engine edit-on Pro
UserLexicon1=userdic.tlx,url,t
UserLexicon2=correct.tlx,url,t
MainLexicon1=ssceam.tlx,url,t
MainLexicon2=ssceam2.clx,url,c
REPORT_UNCAPPED_OPT=false
IGNORE_CAPPED_WORD_OPT=false
SPLIT_WORDS_OPT=false
IGNORE_NON_ALPHA_WORD_OPT=true
REPORT_MIXED_CASE_OPT=true
REPORT_DOUBLED_WORD_OPT=true
IGNORE_ALL_CAPS_WORD_OPT=false
ALLOW_ACCENTED_CAPS_OPT=true
MinSuggestDepth=30
SPLIT_HYPHENATED_WORDS_OPT=true
Suggestions=typographical
IGNORE_MIXED_CASE_OPT=false
STRIP_POSSESSIVES_OPT=true
REPORT_MIXED_DIGITS_OPT=true
SUGGEST_SPLIT_WORDS_OPT=true
IGNORE_DOMAIN_NAMES_OPT=false
IGNORE_MIXED_DIGITS_OPT=false
```

```
SPLIT_CONTRACTED_WORDS_OPT=false
Comparator=Typographical
CASE_SENSITIVE_OPT=true
```

4.9 Adding Words to the Spelling Checker

To activate the “Add Word to spelling checker” feature in edit-on Pro, please complete the following steps :

1. Set the ADDSPELLCHECKWORDURL parameter in the configuration file

```
...
<spellingcheckers>
  ...
  <addspellcheckwordurl url="http://www.yourdomain.com/eopro/AddWord.jsp" />
  ...
</spellingcheckers>
...
```

Set the “url” attribute to the URL address where the Server-side script (CGI/ASP/PHP/JSP file) is available. The CGI/ASP/PHP/JSP script will be responsible for appending the new word to the corresponding text lexicon file.

The following listing shows a sample of a JSP script that adds words to the corresponding lexicon file:

```
<%@ page import="java.io.*" %>
<html>
<head>
<title>Add Word</title>
</head>
<body>
<%
  String language = request.getParameter("LANG");

  String strFile = "D:\\dictionary-directory\\" ;
  if (language.equalsIgnoreCase("US English"))
    strFile = strFile + "userdicam.tlx";
  else if (language.equalsIgnoreCase("German"))
    strFile = strFile + "userdicge.tlx";

  File fileDict = new File (strFile);
  RandomAccessFile dict = new RandomAccessFile(fileDict, "rw");

  boolean line = true;
  String strAdded = " ";
  strAdded="\r\n"+request.getParameter("WORD");
  while (line){

    String s = dict.readLine() ;
    if (s == null){

      dict.writeBytes(strAdded);
      break;
    }
  }
  dict.close();
  %>
</body>
</html>
```

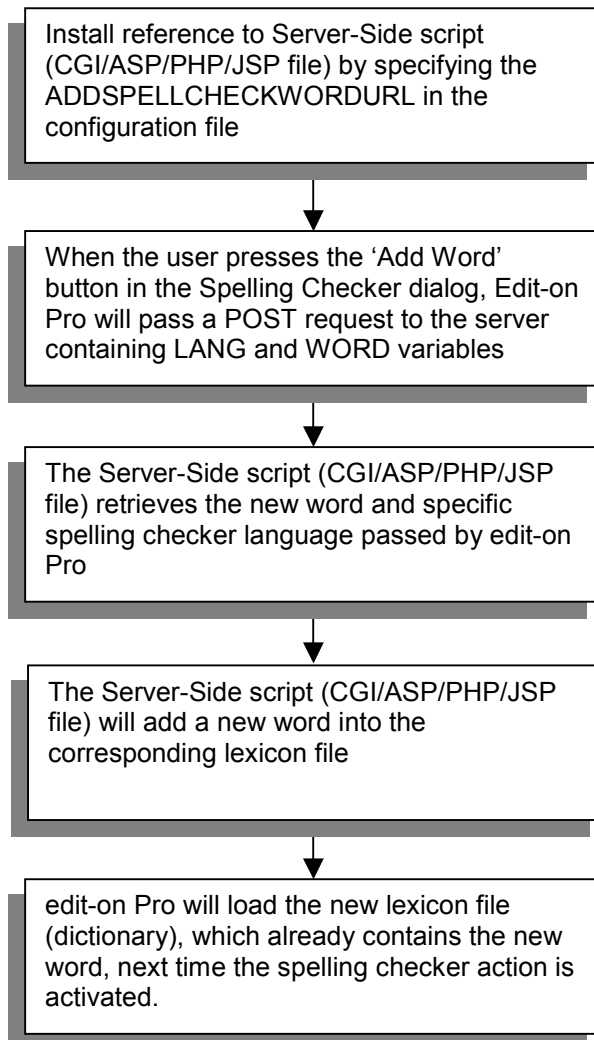
Note:

The Variable LANG will contain the spell checker’s active language, and the variable WORD will contain the new word to add to the dictionary.

2. When the user presses the ‘Add Word’ button in the Spelling Checker dialog, edit-on Pro will pass the value of the variables LANG and WORD to the server-side script (CGI/ASP/PHP/JSP file) as a POST request

3. The server-side script will add the value of variable WORD into the lexicon file specified by the variable LANG.
4. edit-on Pro will load the new lexicon file (dictionary) with the new word the next time the spelling checker action is invoked.

The following diagram describes the steps to add a word to the spelling checker:



5 LOCALE Setting

5.1 Changing the LOCALE

The LOCALE parameter can be used to change the language of the applet's user interface. The parameter's value string is the combination of a language code and a country code:

xx_YY

xx = language code (ISO 639)

YY = country code (ISO 3166)

Currently supported LOCALE values and user interface languages are:

en_US : English (U.S.)

de_DE : German

fr_FR : French

es_ES : Spanish

Note: The LOCALE setting has no influence on the language setting used by the spelling checker.

5.2 Custom Localization

The localization setting for edit-on Pro can be customized using the locale_xx_YY.xml file, where xx_YY is the LOCALE setting.

The locale_xx_YY.xml file contains the information used by edit-on Pro to localize its language setting. Following is the layout syntax for the localization information:

```
<locale>
  <product>edit-on Pro</product>
  <version>VERSION_ID</version>
  <language>xx_YY</language>
<localestrings>
<string id="ID_TEXT">Localization Information</string>
  </localestrings>
</locale>
```

- The 'VERSION_ID' is the version of edit-on Pro.
- The 'xx_YY' is the language code used in the localization file. This complies the locale setting.
- The 'ID_TEXT' is the id used by edit-on Pro to browse through the xml file and extract the localization information. The ID_TEXT should not be changed, because it is as used by edit-on Pro as an ID to browse through the xml file and extract the right localization information. The modification of the ID_TEXT will result in errors.
- The 'Localization Information' is the information that will be extracted by edit-on Pro for its language localization once an id match is found.
- If you have to use special characters e.g. ä,ö,ü,é,â etc. please use the HTML entities like ä, ´ etc.

5.3 Sample Custom Localization

```
<locale>
  <product>edit-on Pro</product>
  <version>3.1.180.0</version>
  <language>en_US</language>
```

```

<localestrings>
  <string id="PANE_WYSIWYG">WYSIWYG</string>
  <string id="PANE_SOURCEVIEW">Source View</string>
  <string id="STATUS_APP_READY">Ready</string>
  <string id="STATUS_APP_LOADING">Loading</string>
  <string id="STATUS_APP_EXPORT">Export</string>
  <string id="STATUS_EDIT_INSERT">Insert</string>
  <string id="STATUS_EDIT_OVERWRITE">Overwrite</string>
  <string id="INSERT_HYPERLINK">Insert Hyperlink</string>
  <string id="INSERT_HTML_FILE">Insert HTML File</string>
  <string id="INSERT_HTML_AT_CURRENT_POSITION">Insert HTML at current
Position</string>
  <string id="EDIT_HYPERLINK">Edit Hyperlink</string>
  <string id="EDIT_TAG">Edit Tag</string>
  <string id="WARNING_EDIT_UNKNOWN_TAG">HTML Markup to insert (will not be checked
for correctness)</string>
  <string id="TYPE_THE_URL_TARGET">Type the URL target</string>
  <string id="URL">URL</string>
  <string id="HREF">Href</string>
  <string id="TARGET">Target</string>
  <string id="SOURCE">Source</string>
  <string id="ALT_IMAGE">Alt-String</string>
  <string id="BORDER_IMAGE">Border</string>
  <string id="SIZE">Size</string>
  <string id="BTN_OK">Ok</string>
  <string id="BTN_CANCEL">Cancel</string>
  <string id="BTN_REMOVELINK">Remove link</string>
  <string id="INSERT_IMAGE">Insert Image</string>
  <string id="EDIT_IMAGE">Edit Image</string>
  <string id="EDIT_IMAGE_PROPERTIES">Edit Image Properties</string>
  <string id="INSERT_SYMBOL">Insert Symbol</string>
  <string id="INSERT">Insert</string>
  <string id="CLOSE">Close</string>
  <string id="OPEN">Open</string>
  <string id="SELECT_FROM_LOCAL_HD">Select from local HD</string>
  <string id="SELECT_FROM_URL_ADDRESS">Select from URL Address</string>
  <string id="OPEN_HTML_FILE">Open HTML File</string>
  <string id="_SELF">Same Frame</string>
  <string id="_TOP">Whole Page</string>
  <string id="_PARENT">Parent Frame</string>
  <string id="_BLANK">New Window</string>
</localestrings>
</locale>

```

6 Troubleshooting Tips

1. edit-on Pro load fails for firewall authentication when using JRE 1.3.0

Java Applets fail to load for firewall authentication when using JRE 1.3.0. This is a known issue in JRE 1.3.0. It is recommended that you upgrade to JRE 1.4.1.

2. Accessing Pop-Up Menus in MacOS

Various Pop-Up Menus within edit-on Pro can be accessed under MacOS by holding the control button, and then clicking the mouse button inside certain areas of the edit-on Pro WYSIWYG view.

3. Using JavaScript in Browsers that use Java Plug In (Microsoft Internet Explorer 6)

To enable JavaScript in Browsers that use a Java Plug In, like Microsoft Internet Explorer 6, make sure you specify the parameter `scriptable` and set its value to `true` in the applet declaration.

4. Using Event Handling with JavaScript

To enable Event Handling with JavaScript, please make sure that you set the `MAYSCRIPT` attribute in the applet tag declaration. See “Event Handling with JavaScript”.

5. How to address a specific applet in a multiple applet environment to be run in Event Handling with JavaScript

To address a specific applet in a multiple applet environment to be run in Event Handling with JavaScript, please make sure you add a function parameter in the JavaScript event handler. This parameter can be used to refer to a specific applet that calls the event handler. See “Event Handling with JavaScript”.

6. How to check the document's tree structure (Document Object Model)?

The user can check the document's tree structure (Document Object Model) using three shortcuts:

- Pressing Ctrl+Alt+P at the same time:

Prints out the document's tree structure in the Java Console.

- Pressing Ctrl+Shift+P at the same time:

Prints out the document's tree structure and the sequence of nodes for each individual character in the document in the Java Console.

- Pressing Shift+Alt+P at the same time:

Prints out the document's tree structure and the list of each individual paragraph's information in the Java Console. The information contains the start offset of the paragraph, the end offset of the paragraph and the node of the paragraph.

10. The JavaScript functions specified for ONEDITORLOADED/ONDATALOADED are not called. My platform supports LiveConnect and JavaScript is enabled.

Make sure to include the `MAYSCRIPT` attribute in your `<applet>` tag. Otherwise the applet is not allowed to call JavaScript functions in your page:

```
<APPLET code="com.realobjects.eop.applet.EditorApplet" archive="edit-on-pro-signed.jar"
height=400 width=750 name="HTMLEditor" MAYSCRIPT>...
```

11. We are running Mac OS 9 and the Internet Explorer crashes after loading the applet several times.

First, make sure the Internet Explorer is completely closed. Then, open the folder where the Internet Explorer is installed and tions in your page and hold "ctrl" and click on the IE icon. Next, select "Information -> Memory" from the upcoming context menu.

Then change the values for minimum and preferred memory usage to "32000". With these memory settings, you should be able to open edit-on Pro up to 60 times within the same window of Internet Explorer without getting the message that IE is running low on memory. Make sure that "Virtual Memory" is enabled on your Mac. 128 MB RAM is strongly recommended.