

RealObjects edit-on Pro Integration Manual

Version 5.3 (5.3.258.337)

Revision: 2014-01-08

© 2000-2014 RealObjects GmbH Saarbrücken, Germany

Introduction

edit-on® Pro is a cross-platform WYSIWYG XHTML/XML editor as a Java applet, allowing individuals and teams to update, create, and publish web content to any Internet or Intranet site. The editor has an easy-to-use, intuitive user interface which provides word processor-like and XML editor-like features to web based applications, empowering non-technical users to become content contributors without knowing XHTML, XML or other cryptic markup languages.

Configured to the precise editing functionality required within a web based application, edit-on Pro allows content authoring while maintaining corporate design and site standards for style, layout and code, thus allowing control of site content and integrity. RealObjects edit-on Pro creates valid XHTML which guarantees portability, compatibility and interoperability of data with other systems. Thus, content can be easily parsed and automatically transformed using XSLT.

RealObjects edit-on Pro features an extensive API that allows full customization of the look and feel, easy configuration and seamless integration into web based applications. The functionality includes CSS support, DTD based validation and definition of custom XML tags, CSS based read-only elements and areas, real-time spell checking – as you type - including 17 dictionaries, localizable multi-language support, table support, cut and paste, drag and drop, context-sensitive menus, fully customizable toolbar, pull down menu and keyboard shortcuts, real-time character counting and limiting.

A tree view allows the navigation within a document's structure to edit the attributes of all elements with DTD guidance. The editor's user interface and XHTML content created are designed to meet the W3C WAI and 508 accessibility guidelines. The Java Accessibility Bridge and common Windows screen readers are supported. RealObjects edit-on Pro is compatible with the most common Web browsers (Internet Explorer, Mozilla, Apple Safari etc.) and client OS, such as Windows, Mac OS X, Linux and Solaris.

1. Basic Integration

1.1 Integrating edit-on Pro in a Web Page

edit-on Pro can easily be integrated into a Web page using JavaScript only. There are two basic steps to perform when integrating edit-on Pro:

- Inserting the editor into a web page
- Importing and exporting content from the editor

1.1.1 Inserting the Editor Into a Web Page

edit-on Pro comes with a comprehensive JavaScript API designed to allow you to configure and integrate the editor into a web site. In the first step, we will demonstrate how to simply load the editor within a web page without any regards to the loading of content or more advanced integration requirements.

The JavaScript API for edit-on Pro basically allows you to create a JavaScript object containing the editor. You can manipulate the editor's configuration by adding and modifying the objects properties before it is finally loaded.

To use the JavaScript API, you should import the "editonpro.js" JavaScript library into the page that will use edit-on Pro.

Example: Importing the edit-on Pro JavaScript library

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

First, we will create an instance of the editor by instantiating the editor's constructor class `editOnPro`.

Example: Creating an instance of edit-on Pro

```
<script type="text/javascript" src="editonpro.js" />
<script type="text/javascript">
eop = new editOnPro(725, 350, "myEditor", "myId", "eop");
</script>
```

We just created an instance of the editor. There are only a few steps until we can load the editor with its default configuration:

Set the editor's codebase. The editor will always be loaded from the codebase. In addition, all configuration files will be loaded from the codebase or from a path relative to the codebase unless specified otherwise.

Specify the editor configuration files. The configuration of the editor will be parsed from these files.

Load the editor. Once the editor has been configured, its `loadEditor` method can be called to insert the editor at the location where it is called.

Example: Setting the codebase and loading the editor

```
eop.setCodebase(".");
eop.setUIConfigURL("uiconfig.xml");
eop.setConfigURL("config.xml");
eop.loadEditor();
```

We now have integrated edit-on Pro into a web page.

1.1.2 Importing and Exporting Content from the Editor

After we learnt how to load the editor in the last chapter, we will now see how content can be imported into or exported from edit-on Pro.

There are multiple ways to load content into the editor: it can be inserted after the editor has initialized using one of the JavaScript API functions. The applet can be configured to load content from an URL using an applet property which can either be set using the JavaScript API or by specifying it in the configuration file.

In this example we will focus on the JavaScript API, as it is used most commonly for setting content.

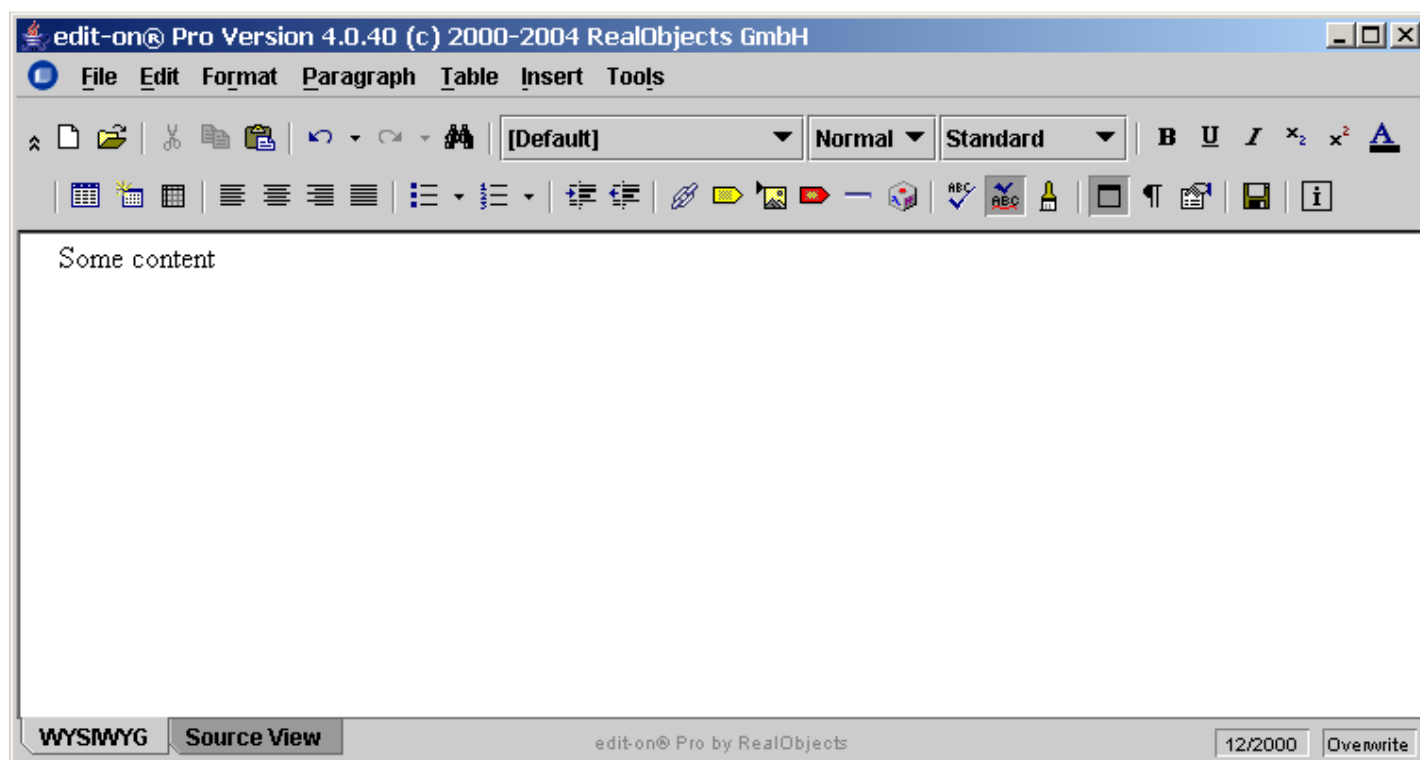
Example: Loading content into the editor using the JavaScript API

```
eop.setHTMLData("<p>Some content</p>");  
eop.pumpEvents();
```

Important:

All JavaScript `set` methods will not be executed immediately, but be queued until the method `pumpEvents()` is executed.

Calling the methods above will insert the string "<p>Some content</p>" into the editor:



We just loaded our first content into the editor. Now we'll see how it can be retrieved, focusing again on the JavaScript API.

Example: Retrieving content from the editor using the JavaScript API

```
myVar = eop.getHTMLData();
```

In the example above, we retrieved the editor's content using the `getHTMLData` method. The complete content of the editor is now stored in the variable `myVar` and can be passed to your application. In a later chapter we will explore how content can be passed to and from the editor using direct HTTP connections instead of the JavaScript API.

1.2 Configuring edit-on Pro

The default configuration and behaviour of the editor can be modified in many ways. For example, if you do not want your users to edit content in the "Source View", you can disable it. If you want to limit the number of characters to be entered, you can do so. All you have to do is to specify the settings you need in the editor's configuration file.

Example: Loading a configuration file

```
eop = new editOnPro(725, 350, "myEditor", "myId", "eop");
eop.setCodebase(".");
eop.setUIConfigURL("uiconfig.xml");
eop.setConfigURL("config.xml");
eop.loadEditor();
```

In the example above, we first instantiate a new editor object called `eop`, then set the editor's codebase and then call the methods `setUIConfigURL` and `setConfigURL`. We'll learn about the `setUIConfigURL` method in the next chapter. The method `setConfigURL` tells the editor where it can find the configuration file. The location of the file may be relative to the applet's codebase. The editor will search for the file at the location specified by the `setConfigURL` method and use the configuration settings specified in this file.

The configuration file is in XML format. The applet can be configured by setting the attributes of specific elements in this file.

For example, if we wanted the user to be able to create documents with 5000 characters at most, we'd look at the `documentcounter` element. Now we can see the `documentcounter` has two attributes, `limit` and `enforcelimit`. We'd also see the attribute `limit` is required while `enforcelimit` is not and has a default value of "false". This means if we use the element `documentcounter` in the configuration file we must indicate the maximum number of characters displayed by the counter. Whether this maximum number is enforced or not is determined by the attribute `enforcelimit`; if this attribute is ignored, the maximum number of characters will only be displayed, but not enforced.

Example: Setting and enforcing the maximum number of characters in a document

```
<documentcounter limit="5000" enforcelimit="true" />
```

The principle described above applies to all elements in the configuration file. Each element has one or more attributes which are either optional or required. Some of the attributes have a default value; attributes having a default value are always optional. In the description of each element, you will see which element is its parent element, and which elements are its children, if any.

Some elements are wrappers for other elements, which are their child elements. For example, the element `fontsizes` is a wrapper for the `fontsize` element. Each `fontsize` element specifies a font size available in the font size selector box, while the element `fontsizes` represents all font sizes available in the selector box.

Example: The elements `fontsizes` and `fontsize`

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

A complete reference of all elements that can be set this way can be found within the [Configuration File](#) chapter.

1.3 Adjusting the User Interface

You have full control over the user interface of edit-on Pro and can influence every aspect of it - from the order of the toolbar buttons and which of them are available to the pull down and context menu. In addition you can even create custom toolbar buttons and menu items which will open your custom classes or execute your custom JavaScript code inside or outside of the editor.

The configuration of the user interface is made within an XML file. To use an UI configuration file you must specify it using the `setUIConfigURL` method. The `setUIConfigURL` must indicate the path to the user interface configuration file. The path may be relative to the editor's codebase.

Example: Loading a user interface configuration file

```
eop = new editOnPro(725, 350, "myEditor", "myId", "eop");
eop.setCodebase(".");
eop.setUIConfigURL("uiconfig.xml");
eop.setConfigURL("config.xml");
eop.loadEditor();
```

Clicking a toolbar button, a context menu or a pull down menu item may have the same the effect. For example, there may be a separate toolbar button, context menu item and pull down menu item for "bold". Thus, we introduced the concept of "actions". Actions represent operations that are executed when elements of the user interface have been selected. Thus in the example above the toolbar button, menu item and context menu item all result in the same operation, i.e. they all cause the selection to become bold. So, we can say each of those buttons triggers the same action. This means the user interface configuration file consists of four different parts:

Elements defining the actions available in the editor. Default actions supported by the editor do not need to be defined explicitly; they can be referred to by any user interface element without need to define them.

Elements defining the toolbar. Each toolbar button refers to the action fired after selection. The toolbar definition influences the available buttons and the order of the buttons in the toolbar.

Elements defining the pull down menu. Each pull down menu item refers to the action it fires if selected. The pull down menu definition influences the available menu items and the order of the pull down menu items.

Elements defining the context menu. Each context menu item refers to the action it fires if selected. The context menu definition influences the available menu items and the order of the context menu items.

1.3.1 Defining Actions

As we said above, the default actions do not need to be defined. However, each action has a number of default properties you may want to modify. You can specify a number of toolbar shortcuts which will fire an action. You can also modify the icon representing an action in the toolbar, context menu and pull down menu.

Example: Defining an action

```
<action id="BOLD" icon="icons/bold.gif">
  <shortcut id="control B" default="true" />
  <shortcut id="control F" default="false" />
</action>
```

In the example above we defined the `BOLD` action. The attribute `id` determines the action we are defining. The `icon` attribute allows you to specify a custom icon for the action. This action also has two child elements called `shortcut`, defining a shortcut triggering the action. The first shortcut will be the default shortcut displayed if the action is used in a pull down menu item while the second shortcut will also trigger the action but is displayed nowhere.

Note:

You will find a list of all action ids available per default in the chapter [Actions](#) .

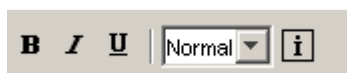
1.3.2 Defining the Toolbar

The toolbar consists of several different elements: toolbar buttons, drop-down (choice) boxes, separators and elements marking the end of a row. Each button and choice box must refer to an action while separators and endrow elements just are empty elements. If we'd like to have a toolbar only allowing the user to use bold, italic and underline formatting and select the font size, we'd define it as follows:

Example: Simple toolbar definition

```
<toolbar>
  <button actionid="BOLD" />
  <button actionid="ITALIC" />
  <button actionid="UNDERLINE" />
  <separator />
  <choice actionid="FONTSIZE" />
</toolbar>
```

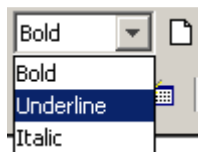
The resulting toolbar will look like this:



As you can see the toolbar only contains the bold, italic and underline buttons, a separator, a choice box and the info button. The info button will always be displayed at the last position in the toolbar.

1.3.2.1 Defining Custom Drop-Down Boxes

Defining a `customchoice` element within your toolbar configuration enables you to group editor actions as well as custom actions within toolbar drop down boxes (choice). Such a drop down box will look like following example:



Example: Simple custom drop down box

```
<toolbar>
  ...
  <customchoice localeid="My customchoice tooltip" fontsize="20">
    <customchoiceitem actionid="BOLD" localeid="Bold"/>
    <customchoiceitem actionid="UNDERLINE" localeid="Underline"/>
    <customchoiceitem actionid="ITALIC" localeid="Italic"/>
  </customchoice>
  ...
</toolbar>
```

Note:

You will find more information on defining custom drop down boxes within chapter [Configuration File](#) . A list of edit-on Pro's default actions can be found in chapter [Actions](#) .

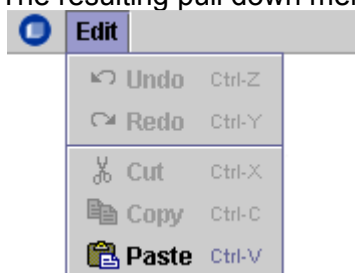
1.3.3 Defining the Pull Down Menu

The pull down menu consists of menu items, separators and menus which can again contain menu items, separators and menus. A menu can be used to group multiple menu items under a common header. For example you may want to define a menu called "Edit" containing the menu items "Undo", "Redo", "Cut", "Copy" and "Paste".

Example: Simple pull down menu

```
<menubar>
  <menu localeid="Edit">
    <menuitem actionid="UNDO"/>
    <menuitem actionid="REDO"/>
    <separator/>
    <menuitem actionid="CUT"/>
    <menuitem actionid="COPY"/>
    <menuitem actionid="PASTE"/>
  </menu>
</menubar>
```

The resulting pull down menu will look like this:



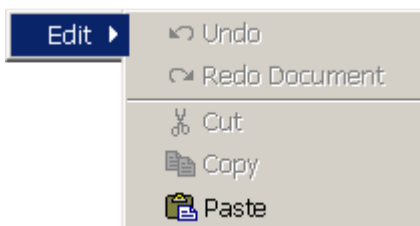
1.3.4 Defining the Context Menu

The definition of the context menu is basically similar to the pull down menu definition, with the exception that the root menu of the contextmenu is the element `contextmenu`. Another difference is that the list of items available in the context menu is built up dynamically depending on context. For example, items allowing the editing of tables are only available within a table.

Example: Simple context menu

```
<contextmenu>
  <menu localeid="Edit">
    <menuitem actionid="UNDO"/>
    <menuitem actionid="REDO"/>
    <separator/>
    <menuitem actionid="CUT"/>
    <menuitem actionid="COPY"/>
    <menuitem actionid="PASTE"/>
  </menu>
</contextmenu>
```

The resulting context menu will look like this:



1.4 Exchanging Data

At the beginning of this chapter, we already saw how to insert / retrieve content using the JavaScript API. We will now focus upon a few things to consider when using this API to insert / retrieve content.

First of all, we'll learn about special considerations to take care of when using the JavaScript API methods described in the chapter [JavaScript API / Java API](#).

Then we'll have a look at the various sources content may come from.

Finally we'll see how legacy content is handled in edit-on Pro.

1.4.1 JavaScript API Considerations

Some JavaScript API functions are not executed immediately after they are called due to their asynchronous nature. These functions will be queued internally and only be executed when the method `pumpEvents()` is called manually. The functions that are concerned are the functions described in the chapter [JavaScript API / Java API](#) excepted all `get` functions which are always executed immediately.

Example: Using JavaScript functions at run-time

```
eop.setImageBase("http://www.internic.com");
eop.setHTMLDataFromURL("http://www.internic.com");
eop.pumpEvents();
```

Note:

The method `loadEditor()` will execute a call to `pumpEvents()` as soon as the editor has fully loaded. Thus all JavaScript API functions called before calling `loadEditor()` will automatically be called once the editor has been fully initialized.

1.4.2 Content Sources

Content may be inserted into the editor from different sources: by loading a document from a URL or by inserting a string from a form field. You also may want to load complete documents or only document fragments. In addition, you can also insert content per drag & drop or cut & paste. Below you will find a few short examples demonstrating how content can be loaded from one of these various sources.

Example: Loading a content fragment using a form field

Form field the content will be loaded from:

```
<form name="myForm">
  <input type="text" name="myContent" value="Some content" />
</form>
```

Inserting the content from this form field into an instance of the editor called `eop` :

```
eop.insertHTMLData(document.myForm.myContent.value);
eop.pumpEvents();
```

Example: Loading content from an URL

```
eop.setHTMLDataFromURL("http://www.w3.org");
eop.pumpEvents();
```

1.4.3 Handling of Legacy Content

By default edit-on Pro will produce valid content according to the XHTML transitional DTD. However, most content on the web or created with Microsoft Word or other office applications will not be valid XHTML.

Thus to assure the editor's interoperability with other applications or legacy data, edit-on Pro will verify the validity of any content inserted and perform a clean-up of the content if it is invalid or not well-formed.

Note:

The clean-up may be disabled or replaced with a custom clean-up process. For more information, see the chapter [The Clean-Up Process](#) .

The editor also allows the validation of documents against specific DTDs. By default, edit-on Pro will validate any content that is loaded against the XHTML transitional DTD. If the inserted data is not valid, it will perform a clean-up of the data and remove any elements and attributes not specified in the DTD that is used.

Note:

For more information on how to validate documents against custom DTDs, see the chapter [Using custom DTDs](#) .

2. Advanced Integration

2.1 Custom actions

There are three types of custom actions you can define:

Custom JavaScript actions. These actions will call a JavaScript function when they are fired.

Custom URL actions. These actions will open a URL when they are fired.

Custom Java actions. These actions will execute a Java class when they are fired., allowing you to execute custom Java classes.

2.1.1 Custom JavaScript actions

A custom JavaScript action calls a JavaScript function when fired. The JavaScript function must be defined in the page loading the editor. For example if we wanted the editor to have a toolbar button inserting the current date, we'd first implement a JavaScript function inserting the current date into the editor when called, then we'd define an action calling this function when it is fired and finally we'd add a button to the toolbar associated with this custom action. Moreover, when the JavaScript function called by the action is fired, it will receive a reference to the applet as first argument, which can then be used to communicate with the applet.

Example: The `insertDate` function

```
function insertDate(applet) {
    now = new Date();
    month = now.getMonth();
    month++;
    applet.insertHTMLData(now.getFullYear()+"-"+month+"-"+now.getDate());
    applet.pumpEvents();
}
```

Example: Defining the custom JavaScript action `INSERTDATE`

```
<actions>
  <actionjs id="INSERTDATE"
            jsfunction="insertDate"
            icon="icons/date.gif">
    <shortcut id="alt D"></shortcut>
  </actionjs>
</actions>
```

Example: Defining a toolbar button calling the custom action `INSERTDATE`

```
<toolbar>
  <button localeid="Insert Date" actionid="INSERTDATE"/>
</toolbar>
```

2.1.2 Custom URL actions

A custom URL action will open a URL when fired. The target frame in which the URL will be opened can be specified optionally.

Example: Defining a custom URL action

```
<actions>
  <actionurl id="ROWEBSITE"
            icon="icons/url.gif"
            url="http://www.realobjects.com"
            target="_new" />
</actions>
```

2.1.3 Custom Java actions

A custom Java action will call a custom Java class when fired. The class must be available within the editor's classpath. You can ensure this by placing the class in a Java Archive (.jar) and loading this archive using the JavaScript API method `setArchive`. For details, see the chapter [JavaScript API](#).

Example: Defining a custom Java action

```
<action id="TEMPLATEPICKER"
  class="com.realobjects.customdialogs.templatepicker.TemplatePicker"
  icon="icons/templatepicker.gif" />
```

2.2 edit-on Pro CSS Extensions

edit-on Pro allows you to define read-only areas and read-only elements using CSS. Style sheets are also used to dynamically populate the context menu, and to specify whether custom elements should always be visible and how they are represented.

Only custom elements can have the custom style properties "ro-readonlycontent" and "ro-readonlyelement". Any other elements (such as "`<p></p>`") are not affected by these style properties.

2.2.1 Read-Only Content

Read-only content parts of the document inside the editor are write protected. Any editing actions such as typing, deleting, changing the font size etc. executed on read-only content or on a selection overlapping it is ignored. You can declare the content inside an element as read-only area by using the `ro-readonlycontent` style property. Elements with the `ro-readonlycontent` style property are automatically also treated as read-only elements.

Example: Declaring the content inside an element as read-only content

Style definition:

```
.readonlycontent { ro-readonlycontent:true; }
```

Using this style in the document:

```
<mycustomtag class="readonlycontent">
  This content can not be modified.
</mycustomtag>
```

In the example above, the content within any element using the style class "readonly" is considered as read-only content.

2.2.2 Read-Only Elements

Read-only elements can not be removed from a document. However, the content within a read-only element may be modified. For example when your users are not allowed to remove certain parts of the document, such as news headlines, but are allowed to edit them, you could employ read-only elements.

Example: Declaring read-only elements

Style definition:

```
customtag { ro-readonlyelement:true; }
```

Using this style in the document:

```
<customtag>
  This element can not be removed.
</customtag>
```

2.2.3 Dynamically Populating the Context Menu Using CSS

All elements available in the context menu are defined in the `contextmenu` section of the user interface configuration file. Whether or not a specific context menu item is available for a certain element is handled using style sheets. The `TABLEPROPERTIES` action for example is only available in the context menu when the cursor is inside a table because this action is only declared for the element `table`.

Example: Enabling the `TABLEPROPERTIES` action in the context menu

The `TABLEPROPERTIES` action must be defined in the context menu:

```
<menu localeid="Table">
  <menuitem actionid="INSERTROW"/>
  <menuitem actionid="INSERTCOLUMN"/>
  <separator/>
  <menuitem actionid="CELLPROPERTIES"/>
  <menuitem actionid="ROWPROPERTIES"/>
  <menuitem actionid="COLUMNPROPERTIES"/>
  <menuitem actionid="TABLEPROPERTIES"/>
</menu>
```

Linking this action to the table element using the style `ro-action`:

```
table { ro-action:TABLEPROPERTIES; }
```

As you can see, the `table` element is the only element having the style `ro-action:TABLEPROPERTIES`. Thus, the `TABLEPROPERTIES` action is only available in the context menu when the caret is inside a table.

2.2.4 Controlling the display of Custom Elements

By default custom elements not recognized by the editor (e.g. elements added using a custom DTD) will not be rendered in WYSIWYG mode. They will only be displayed when the `SHOWNALLCHAR` action is fired. However you can use the style `ro-alwaysshowtags` causing edit-on Pro to always display elements having this style. You can even specify images to be used to display the opening and closing tags of unrecognized elements using the styles `ro-starttagicon` respectively `ro-endtagicon`. For example, if we wanted the editor to display text input fields differently, we'd define styles as follows:

Example: Using Styles Controlling the Display of Elements

```
input[type=text] {
  ro-alwaysshowtags:true;
  ro-starttagicon:"icons/start.gif";
  ro-endtagicon:"icons/end.gif";
}
```

2.3 Direct HTTP Connection API

The direct HTTP connection API can be used to interface with the applet instead of the JavaScript API. The properties `sethtmldataurl` and `gethtmldataurl` allow you to respectively import or export content. They may be defined in the configuration file or using the JavaScript API method `setParam` before initialization of the editor.

In any case, edit-on Pro will perform a request to the URL specified by the `sethtmldataurl` property, and load the data from the document specified by the URL into the editor.

Example: Setting `sethtmldataurl` and `gethtmldataurl`

Setting `sethtmldataurl` using the JavaScript API:

```
eop.setHTMLDataFromURL("http://www.w3.org");
```

Setting `sethtmldataurl` in the configuration file:

```
<directhttp sethtmldataurl="http://www.w3.org" />
```

The editor will send its content in a form field called `HTMLDATA` to the URL specified by the `gethtmldataurl` property when the `SAVE` action is triggered. The request made by the editor when sending the content will be a HTTP POST request which will be application/x-www-form-urlencoded. The server application must reply `<result>OK</result>` to the applet when receiving this request.

Example: Sample `gethtmldataurl` handlers

Sample `gethtmldataurl` handler in 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 in PHP:

```
<?
$myfile = fopen("save.htm","wb");
fwrite($myfile,stripslashes($_POST["HTMLDATA"]));
fclose($myfile);
?>
<result>OK</result>
```

Sample `gethtmldataurl` in 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();
}
%>
```

If an error occurred when saving the content, you can return an error message instead of "OK" in the server-side script handling the save. If anything but "OK" is returned by your HTTP POST handler, you can retrieve the string that was returned instead using the JavaScript API method `getPostErrorMessage`.

If neither "OK" nor an error message is returned by your script, `getPostErrorMessage` returns `null`. Example:

```
function onUploadFinished() {
    alert(eop.getPostErrorMessage());
}
eop.uploadToRepository("UPLOADIMAGES","onUploadFinished");
```

2.4 Localizing edit-on Pro

The user interface of edit-on Pro is completely localizable. The user interface languages available by default are English (U.S.), German, French and Spanish. The `locale` property is used to select the language used by the editor's user interface.

2.4.1 The `locale` Property

The `locale` property consists of a combination of a language code and a country code `xx_YY` where:

`xx` = language code (ISO 639)

`YY` = country code (ISO 3166)

Note:

The locale setting has no influence on the language used by the spell checker.

2.4.2 Custom Localization

If you want to customize the localization of the editor, you can use the `localeurl` property to indicate the location of a custom XML locale file. The editor will use this file to localize the user interface. The locale file is an XML file mapping the strings used in dialogs and the `localeid` attributes of the elements defined in the user interface configuration file to the localized strings defined in the locale file.

The locale file must contain the following information:

```
<locale>
  <product>edit-on Pro</product>
  <version>VERSION</version>
  <language>xx_YY</language>
  <localestrings>
    <string id="ID">Localization information</string>
    .
    .
    .
  </localestrings>
</locale>
```

`VERSION` is the version of edit-on Pro

`xx_YY` is the language code used in the localization file. This complies to the format used by the locale setting.

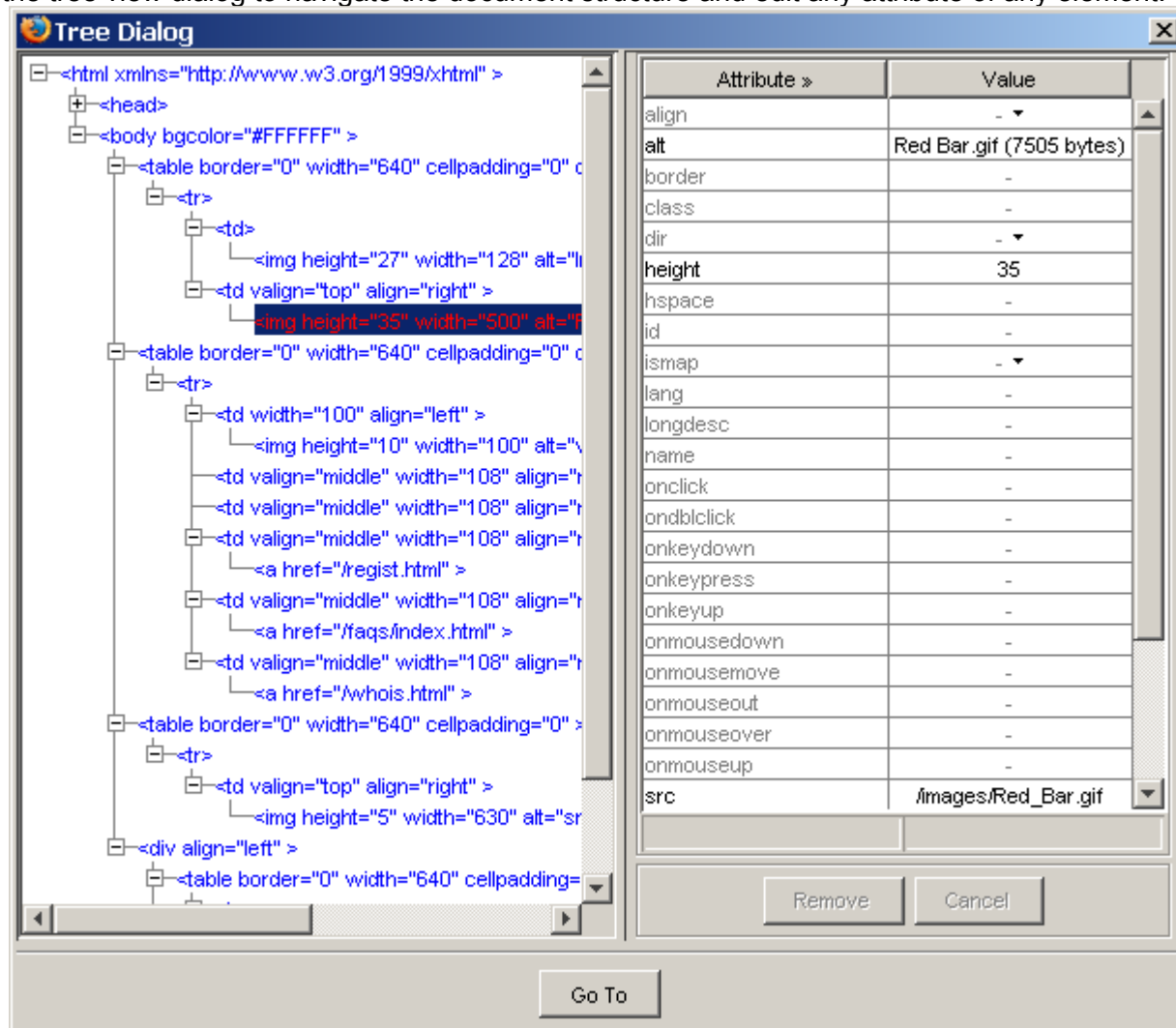
`ID` is the ID of the user interface element that will be mapped to this localized string.

`Localization information` is the string the user interface element with the corresponding ID is mapped to.

2.5 Accessibility

edit-on Pro supports the creation of accessible content according to the Section 508 guidelines and the Web Content Accessibility Guidelines by the W3C Web Accessibility Initiative (WAI). Furthermore, the user interface of edit-on Pro was developed with attention to Section 508 and the Authoring Tool Accessibility Guidelines.

edit-on Pro has several features allowing you to ensure the accessibility of the created content. You can use the tree view dialog to navigate the document structure and edit any attribute of any element.



In addition, if you use the XHTML Transitional DTD and the editor's DTD validation feature, attributes necessary for the creation of accessible content will be required by the editor. For more details about the creation of accessible content, please see the [Web Content Accessibility Guidelines](#) by the WAI .

2.6 The Clean-Up Process

edit-on Pro will automatically validate any content inserted against the XHTML transitional DTD. Depending on the configuration, the editor will perform a clean-up on the document and remove any attributes and elements not allowed according to the DocType set in the document. In this section we will see which types of clean-up processes are available when they will be invoked and how custom clean-up processes can be configured.

2.6.1 Types of Clean-Up Processes

Four types of clean-up processes are invoked depending on the situation:

- while importing content into the editor
- while pasting content from the clipboard into the editor
- while pasting content from the clipboard using the `PASTEWITHFILTER` action
- while exporting content

Each of these clean-up processes can be overridden by custom clean-up processes. A custom clean-up can occur either on the server using a server side scripting language or on the client using custom Java classes. Default clean-up processes are configured in the configuration using the element `cleanupprocess`. For details see the chapter [Configuration File](#) .

2.6.2 Configuring Custom Clean-Up Processes

2.6.2.1 Customizing the Client Side Clean-Up Process

edit-on Pro allows you to fully customize the client side clean process by providing an own implementation of the process. To do so you can write an own Java class extending the class

`com.realobjects.eop.bean.custom.CleanUpProcess`. For a description of the methods available in this class, see the chapter [Methods and Properties of the Client Side Clean-Up Base Class](#).

The example below shows how to can configure a custom clean-up process in the configuration file:

Example: Configuring a custom client side clean-up process in the configuration file

```
<cleanupprocess  
  class="com.mycompany.cleauextension.MyCleanUp" />
```

The following code snippet shows a skeleton of a custom clean-up class extending the `com.realobjects.eop.bean.custom.CleanUpProcess` class. All clean-up methods available are provided in this example without an implementation. It is possible to override each of these methods individually, i.e. you do not have to override all four methods while implementing a custom clean-up process.

Example: Extending the `com.realobjects.eop.bean.custom.CleanUpProcess` class

```
package com.mycompany.cleanuextension;
import com.realobjects.eop.bean.custom.CleanUpProcess;
/**...*/
public class CleanUp
    extends CleanUpProcess {
    /**...*/
    /**
     * Cleans up the raw data received when importing content
     * and returns the cleaned data if no exception occurs.
     * This method should be overridden for a
     * custom implementation of the clean-up process.
     * @param strRawData raw data to be cleaned
     * @return clean data
     * @throws Exception thrown if clean up process fails
     */
    public String cleanUpOnLoad(String strRawData)
        throws Exception {
        //TODO: implement your clean-up process
    }
    /**
     * Cleans the raw data received from the clipboard when the
     * PASTEWITHFILTER action is fired and returns the cleaned
     * data as if no exception occurs.
     * This method should be overridden for a custom
     * implementation of the clean-up process.
     * @param strRawData raw data to be cleaned up
     * @return clean data
     * @throws Exception thrown if clean up process fails
     */
    public String cleanUpClipboardWithFilter(String strRawData)
        throws Exception {
        //TODO: implement your clean-up process
    }
    /**
     * Cleans the raw data received from the clipboard and
     * returns the cleaned data if no exception occurs.
     * This method should be overridden for a custom
     * implementation of the clean-up process.
     * @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
    }
    /**
     * Cleans the raw data received from the editor when saving
     * and returns the cleaned data if no exception occurs.
     * This method should be overridden for a custom
     * implementation of the clean-up process.
     * @param strRawData raw data to be cleaned up
     * @return clean data
     * @throws Exception thrown if clean up process fails
     */
    public String cleanUpOnSave(String strRawData)
        throws Exception {
        //TODO: implement your clean-up process
    }
}
```

2.6.2.2 Customizing a Server Side Clean-Up Process

Instead of customizing the clean-up process on the client, you can perform a clean-up on the server. In this case you have to modify your configuration file appropriately, i.e. you have to determine the location of the script handling the clean-up using the `url` attribute of this element.

Example: Configuring a custom server-side clean-up process in the configuration file

```
<cleanupprocess  
  url="http://myserver.com/customCleanUp.jsp" />
```

Once you have configured a clean-up process, the clean-up information will be sent to the server by a POST request. For details about the fields that are sent to the server please see the chapter [Form Fields Sent to the Server by the Server Side Clean-Up Process](#). On the server you have to parse the content to be cleaned and return the cleaned content to the applet.

Below you find a code snippet showing a skeleton of a sample JSP server side clean-up process handler.

Example: Sample server-side clean-up process script

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

<%!
/**
 * Cleans up the raw data received when importing content
 * and returns the cleaned data if no exception occurs.
 * This method should be overridden for a
 * custom implementation of the clean-up process.
 * @param strRawData raw data to be cleaned
 * @return clean data
 * @throws Exception thrown if clean up process fails
 */
public String cleanUp(String strRawData)
                        throws Exception {
    //TODO: implement your clean-up process
    return strResult;
}

/**
 * Cleans up the raw data received when importing content
 * and returns the cleaned data if no exception occurs.
 * This method should be overridden for a
 * custom implementation of the clean-up process.
 * @param strRawData raw data to be cleaned
 * @return clean data
 * @throws Exception thrown if clean up process fails
 */
public String cleanUpWithFilter(String strRawData)
                                throws Exception {
    //TODO: implement your clean-up process
    return strResult;
}
%>
<%
String content = request.getParameter("HTMLDATA");
String type = request.getParameter("TYPE");
String cleanedData = null;
PrintWriter res = response.getWriter();

try {
    if ( "cleanuponsave".equalsIgnoreCase(type) )
        cleanedData = cleanUp(content);

    if ( "cleanuponload".equalsIgnoreCase(type) )
        cleanedData = cleanUp(content);

    if ( "cleanupclipboard".equalsIgnoreCase(type) )
        cleanedData = cleanUp(content);

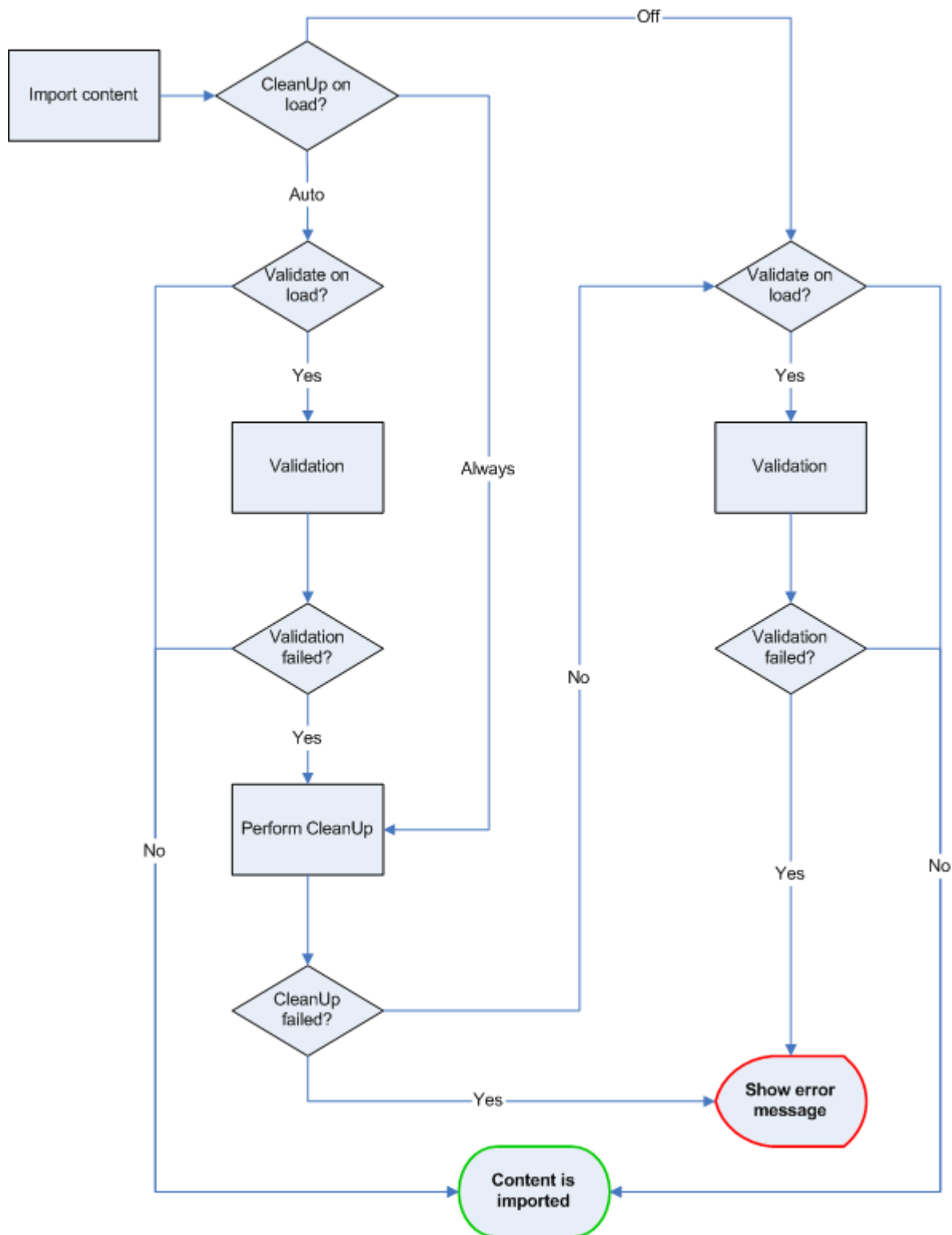
    if ( "cleanclipboardwithfilter".equalsIgnoreCase(type) )
        cleanedData = cleanUpWithFilter(content);
} catch ( Exception e ) {
    e.printStackTrace();
}

if ( cleanedData != null )
    res.write(cleanedData);
else
    res.write("An error occurred during the clean up process.");

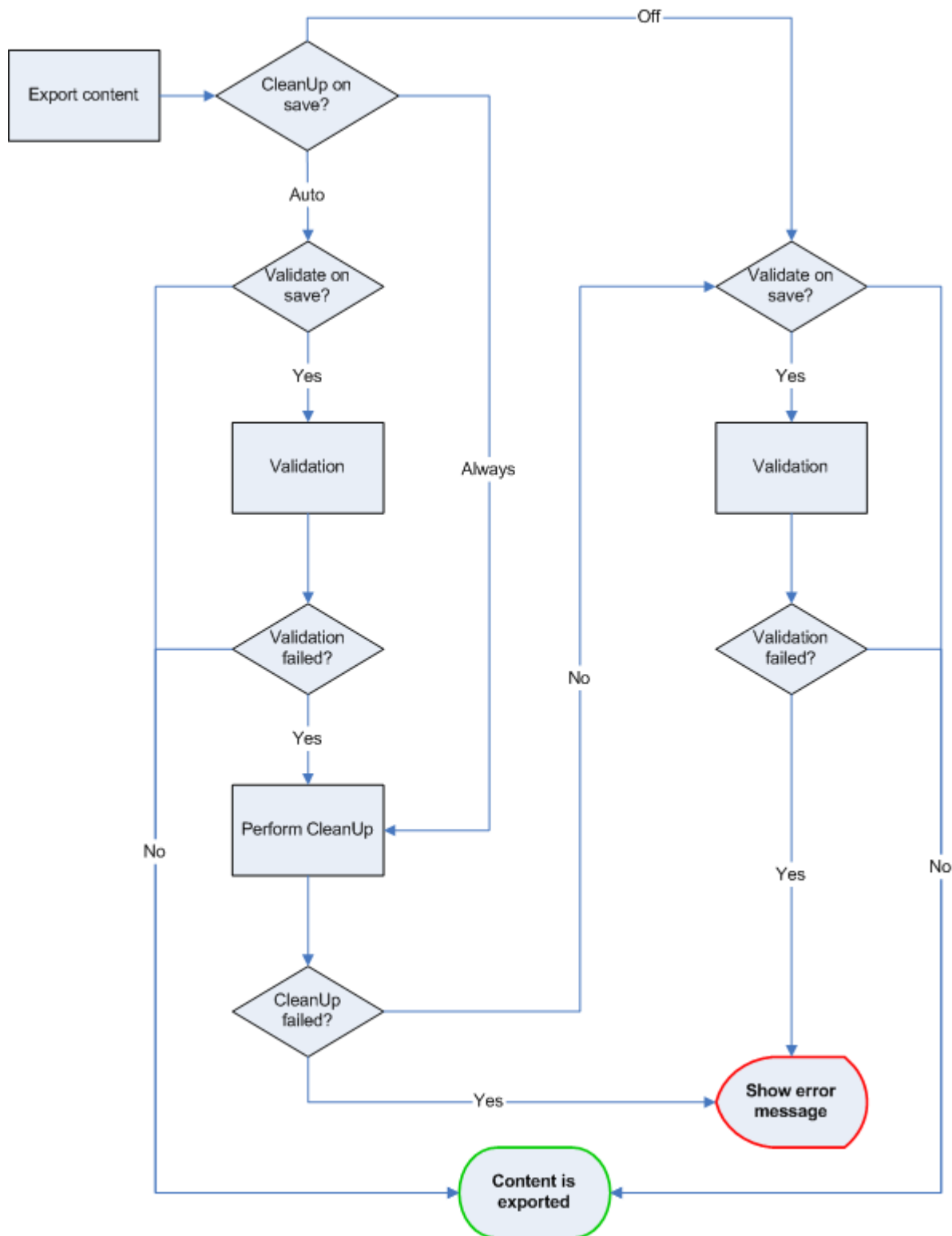
res.flush();
%>
```

2.6.3 Clean-up Process Flow Charts

The following flow charts show when the clean-up process is called while importing or exporting content.

2.6.3.1 Clean-Up Process when Content is Imported

2.6.3.2 Clean-Up Process when Content is Exported



2.6.4 Configuring Drag & Drop

2.6.4.1 Available Drag & Drop Handlers

When content is dragged and dropped inside edit-on Pro, it will trigger the `DRAGNDROP` action by default. Inserting content in the editor with the `DRAGNDROP` action as drag & drop handler corresponds to pasting content using the `PASTE` action as paste handler, i.e. this allows to insert formatted content, which will only be cleaned by the clean-up process if necessary. The other available drag & drop handlers are `DRAGNDROPWITHFILTER` and `DRAGNDROPPLAINTEXT`. `DRAGNDROPWITHFILTER` will remove all inline styles from the

content that is inserted via drag & drop, while `DRAGNDROPPLAINTEXT` will insert it as plain text without any formatting. The drag & drop handler is configured in the "uiconfig.xml".

Example: Configuring the drag & drop handler

```
<actions>
  <dragndropaction actionid="DRAGNDROPPLAINTEXT" />
</actions>
```

2.6.4.2 Customizing Drag & Drop

Instead of using the default drag & drop handlers available in the editor, you can in fact use any action as drag & drop handler - even custom actions. To do so, you need to specify the action that should be used as drag & drop handler in the "uiconfig.xml" (see example above). This action will be triggered whenever content is dragged & dropped in the editor. For example, you could fire the `BOLD` action when content is dragged and dropped in the editor. Note this would not cause the content that was dragged & dropped in the editor to be inserted as bold content, it would only trigger the `BOLD` action as described in the chapter [Actions](#).

If the action that is specified as drag & drop handler is custom Java action, the following data will be passed to the custom action:

The transferable data passed in `DropTargetDropEvent`

The action mode (either move or copy, depending on whether the control key was pressed down when dragging the content)

The start location of the Drag source

The stop location of the Drag source

The target location of the Drop target

These properties will be available as the following key-value pairs in the `EOPDATA` Hashtable using the `getKeys` method (instead of the properties normally available in a custom Java action):

Keys available in EOPDATA for custom drag & drop handlers

transferable:

The Transferable data passed in `DropTargetDropEvent`, may contain a string, a List of the file that was dragged, or a URL, depending on the data flavor that was the Drag source. The supported data flavors are: `text/html` (data is dragged into the editor), `application-x-java-file-list` (a file is opened from the file system via drag & drop), `application-x-java-url` (a link is inserted in the editor).

action:

An Integer specifying the action type passed in `DropTargetDropEvent`. `DropTargetDropEvent` will either pass `DnDConstants.ACTION_COPY` or `DnDConstants.ACTION_MOVE` depending on whether the content was copied or moved into the editor via drag & drop.

sourceStart:

Integer describing the start offset of the Drag source.

sourceEnd:

Integer describing the end offset of the Drag source.

targetLocation:

Integer describing offset of the Drop target.

Note:

For details about custom actions and custom dialogs, see the chapter [Creating Custom Dialogs](#).

2.7 Using Custom DTDs

2.7.1 Specifying a Custom DocType

By default edit-on Pro will validate documents loaded against the XHTML transitional DTD. If the document is invalid, it will perform a clean-up on the document and remove any elements and attributes not allowed in the DTD specified. If you want to use elements and attributes not specified in the XHTML transitional DTD, but still like to validate the document in the editor against a DTD, you can do this by either loading a document or a template with a custom DocType into the editor.

Example: Specifying a custom DocType in a document or template

```
<!DOCTYPE mydoctype SYSTEM "mydoctype.dtd">
```

In the example above, we use the DocType `mydoctype`, defined in the DTD `mydoctype.dtd`. Instead of indicating a DTD available on your server, you could of course also specify a public DocType.

Note:

If you specify a custom DTD with a relative path, it will be resolved relative to the code base per default. You can set a base URL for DTDs by using the JavaScript API function `setDTDBase`. If `setDTDBase` is specified, all DTDs with a relative path will be resolved relative to the URL specified by `setDTDBase`. For details, see the chapter [JavaScript Configuration API](#).

Example: Specifying a public DocType in a document or template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The default DocType supported by edit-on Pro is XHTML transitional. Content created using the editor will be valid XHTML transitional by default. Although edit-on Pro can validate documents against other DTDs and clean them if necessary, the editing functionality of the editor will not be modified by the custom DTD i.e. it will still create content according to the XHTML transitional DTD. If for example you were using a custom DTD with no tables allowed, it may still be possible to create tables using the editor. In this case, you have to make sure the table dialogs are deactivated and that tables are removed when the document is saved. You can do so by setting the `validateonsave` attribute of the `documentvalidation` element to `true` in the configuration file and enabling and configuring the clean-up process to remove illegal elements and attributes.

2.7.2 Defining custom tags

Custom tags have to be defined in a custom DTD so they can be used in edit-on Pro. To do so, you can either create your own custom DTD, or use the `xhtml1-transitional-eop.dtd` which is provided in the edit-on Pro installation package. The `xhtml1-transitional-eop.dtd` basically corresponds to the standard XHTML transitional 1.0 DTD, with the exceptions that two entities were added (`eopcustomblock` and `eopcustominline`) and that this DTD references another DTD, `eopcustomelements.dtd`. Now you can define all custom elements and their attributes in `eopcustomelements.dtd` by adding the required elements either to the `eopcustomblock` or `eopcustominline` entity, depending on whether you require inline or block elements or both. Next, add the element and attribute definitions to the DTD.

Example: Sample eopcustomelements.dtd

```
<!ENTITY % eopcustomblock "myblock">
<!ENTITY % eopcustominline "myinline">

<!ELEMENT myblock ANY>
<!ATTLIST myblock id CDATA #IMPLIED name CDATA #REQUIRED >

<!ELEMENT myinline EMPTY>
```

In the example above, we defined a custom block level element called `myblock` and a custom inline element called `myinline`. The element `myblock` has two attributes, `id` and `name`, the latter being required, i.e. it is always inserted with the element `myblock`. The element `myinline` is defined as an empty inline element.

Now all there is left to do is to make sure the editor always validates content against the `xhtml1-transitional-eop.dtd`, which in turn includes the `eopcustomelements.dtd`. This can be done by referencing the `xhtml1-transitional-eop.dtd` in a template loaded by the editor. For details about templates, please see [Template Based Editing](#).

2.8 Template Based Editing

edit-on Pro supports template based editing. You can load a template by using the `templateurl` property or by setting the JavaScript API functions `setTemplate` or `setTemplateURL`. If you specify a template using the `templateurl` property, the template is loaded before any other content is loaded into the editor. In addition, the template will be loaded as base template for any new documents (i.e. every time the `NEW` action is fired).

Example: Specifying a template in the configuration file

```
<templateurl url="mytemplate.xhtml" />
```

2.9 Managing Embedded Files

As files can be inserted into edit-on Pro from various sources such as the local file system and the Internet, the editor provides a file upload mechanism which finds all files in the document not located in the file base and allow the user to upload them either by performing HTTP POST requests to your server or using WebDAV if you have a WebDAV-enabled server.

Note:

The file base can be specified in the configuration file using the `base` element. If this element is specified the editor will search for files with a relative path in a location relative to the `base`. If the element is not specified, the file base defaults to the code base.

2.9.1 Uploading and Inserting Files Using WebDAV

To use this technology to transfer files from and to your server, your web server must support the WebDAV protocol. If your server meets this requirement, you can set up edit-on Pro to allow the uploading of any files not located in the file base to a directory on your web server and the insertion of files into the document being edited directly from your file repository. To do so, you will have to activate WebDAV in the configuration file, and enable the according dialogs to upload and insert files.

Example: Enabling upload and insert dialogs

```
<uiconfig>
...
<toolbar>
...
<button actionid="UPLOADIMAGES" />
<button actionid="UPLOADOBJECTS" />
<button actionid="ULOADDOCUMENT" />
<button actionid="IMAGE" />
<button actionid="OBJECT" />
<button actionid="LINK" />
...
</toolbar>
...
</uiconfig>
```

The example above shows how to activate the actions for the upload and insertion of files. Each of these dialogs handles a different type of files (images, objects, and the document currently being edited). To enable the browsing of the server repository in the corresponding dialogs, the according repository for this file type must be specified in the configuration file. Here is an overview of which file types are used with which actions and of the respective purpose of each setting:

Overview of ACTIONS and their corresponding configuration

Action	Configuration Setting	Purpose
UPLOADIMAGES	<file type="image">	Allows to upload images to the specified repository.
UPLOADOBJECTS	<file type="object">	Allows to upload objects Allows to upload images to the specified repository.
UPLOADDOCUMENT	<file type="document">	Allows to save the document currently being edited to the specified repository.
IMAGE	<file type="image">	Allows to insert images from the WebDAV repository.
OBJECT	<file type="object">	Allows to insert objects from the WebDAV repository.
LINK	<file type="link">	Allows to insert hyperlinks to the files in the specified WebDAV repository.
LOAD	<file type="document">	Allows to load a document from the WebDAV repository.

See below for an example on how to configure a repository for each file type.

Example: Activating WebDAV in the configuration file

```
<filemanagement>
  <file type="image"
    baseurl="http://www.myserver.com/"
    autoupload="true"
    onuploadfinished="uploadStatusReport">
    <webdav url="http://www.myserver.com/repository/images"
      username="user"
      userpassword="mypw"
      debug="true"
      putmethodexpectation="false"
      mimetype="image/gif" />
  </file>
  <file type="object"
    baseurl="http://www.myserver.com/"
    autoupload="true">
    <webdav url="http://www.myserver.com/repository/multimedia"
      username="user"
      userpassword="mypw"
      debug="true"
      putmethodexpectation="false" />
  </file>
  <file type="document"
    baseurl="http://www.myserver.com/">
    <webdav url="http://www.myserver.com/repository/docs"
      username="someotheruser"
      userpassword="someotherpw"
      debug="true"
      putmethodexpectation="false" />
  </file>
</filemanagement>
```

In the example above we added three buttons to the editor's toolbar. They respectively open the "Upload Images" dialog, the "Upload Objects" dialog and the "Upload Document" dialog. Thus, we have now configured the editor to allow the user to upload any images and objects not located in the file base to the location "http://www.myserver.com/repository/images" for images or "http://www.myserver.com/repository/objects". As the file base allows the editor to look up images with relative URLs, it is a good idea to set the file base location to the root of the WebDAV server you are using.

Example: Configuring the file base

```
<config>
  ...
  <base href="http://www.myserver.com/" />
  ...
</config>
```

An additional benefit of using WebDAV is it will activate an directoryadditional pane in the "Insert Image", "Insert Object" and "Insert Hyperlink" dialogs which will let the user insert images or objects directly from the

WebDAV repository and let him browse and place a link to these documents within the "Insert Hyperlink" dialog.

Note:

You can enable a debug mode for WebDAV which logs all actions performed by the WebDAV to the Java console by setting the `debug` attribute of the `webdav` element to `true`.

Important:

The password for the WebDAV connection specified by the `userpassword` attribute may not contain whitespace. Please make sure it does not contain any whitespace or the connection to the WebDAV repository may fail if a password is required.

2.9.2 Uploading Files Using HTTP Connections

If your web server does not support WebDAV, you can upload files per HTTP POST. To activate this upload method, you have to configure the editor accordingly.

Example: Activating the HTTP POST upload mechanism in the configuration file

```
<filemanagement>
  <file type="image" baseurl="http://www.myserver.com/"
    autoupload="true"
    onuploadfinished="uploadStatusReport">
    <http url="http://www.myserver.com/upload.php" />
  </file>
  <file type="object" baseurl="http://www.myserver.com/"
    autoupload="true">
    <http url="http://www.myserver.com/upload.php" />
  </file>
  <file type="document" baseurl="http://www.myserver.com/" >
    <http url="http://www.myserver.com/upload.php" />
  </file>
</filemanagement>
```

As you can see, the configuration is similar to the configuration used for WebDAV, the main difference being that we do not specify repositories using the `webdav` element, but instead specify a server-side script to take care of the file upload.

Once the HTTP POST file upload is activated, all files included in the document can be uploaded using the one of the upload dialogs. The files selected for upload images will then be sequentially posted to the server side script handling the file upload specified by the `url` attribute of the `http` element in the configuration file.

Each file will be contained in a form field called `uploadfile`. The server side script must return "OK" if the file upload succeeded and return "Failed" if it failed. In addition, you can return the new absolute path the file will be saved to. The editor will then modify the path of the embedded file in the document.

Example: Sample server side script handling the image upload (PHP)

```
<?php
$filename = $_FILES["uploadfile"]["name"];
$tmp_name = $_FILES["uploadfile"]["tmp_name"];

if(file_exists($filename)) {
    while(file_exists($filename)) {
        $ext = strrchr($filename, ".");
        $pos = strpos($filename, ".");
        $filename = substr($filename, 0, $pos) . "$" . $ext;
    }
}

if($fp = fopen($tmp_name, "rb")) {
    $file = fread ($fp, filesize($tmp_name));
    fclose($fp);
} else
    die("Failed");

if($fp = fopen($filename, "wb")) {
    fwrite($fp, $file);
    fclose($fp);
} else
    die("Failed");

echo "OK";
echo "|http://yourserver.com/eop4".$filename."|";
?>
```

2.9.3 Using the JavaScript API to Trigger the File Upload

You can use the JavaScript function `uploadToRepository` to trigger the file upload. The `uploadToRepository` function will open the "Upload Images", "Upload Document" or "Upload Document" when it is fired, depending on the file type you selected. The function allows you to specify another JavaScript function as argument, which will receive a hashtable as parameter when the upload is finished. This hashtable will contain status information about each file that was uploaded. It will contain the following info for each file:

The old name of the file

The new name of the uploaded file

The new file path on the server

The status code returned by the server

The type of the upload that was performed (`image` , `object` or `document`)

The event handler specified by `uploadToRepository` will actually receive two arguments: a reference to the applet and the hashtable with upload information.

Example: Using the `uploadToRepository()` function

```
function checkForUpload(obj,state) {
  for (i=0;i<state.length;i++) {
    alert("Upload status:\n"
      +state[i][0]
      +"\n"+state[i][1]
      +"\n"+state[i][2]
      +"\n"+state[i][3]
      +"\n"+state[i][3]
      +"\n"+state[i][4]
      +"\n");
  }
}

eop.uploadToRepository("UPLOADIMAGES","checkForUpload");
```

Note:

Calling the `uploadToRepository` function inside an event queue will halt the event cycle. I.e., if you are calling any other methods that need to be registered in the queue after the `uploadToRepository` function, call `pumpEvents` within the definition of `uploadToRepository`'s event handler.

2.10 Creating Custom Dialogs

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 Overview

In edit-on Pro you can create custom dialogs by overwriting existing dialogs or creating new dialogs. Overwriting an existing dialog is done by overwriting the action linked to it. For example, if you wanted to overwrite the "Insert Link" dialog, you'd need to overwrite the `LINK` action.

2.10.2 Implementing a Custom Dialog

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

Construct a class that extends the abstract class `javax.swing.AbstractAction`.

Override the method `actionPerformed()` of the `java.awt.event.ActionListener` interface.

In the `actionPerformed()` method, build and display your custom user interface and handle the data exchange between your class and edit-on Pro.

Build your class.

Deploy it to your server.

Specify your custom class as a class attribute of the selected action element in the user interface configuration file and hand over your class name.

2.10.2.1 Interfacing with edit-on Pro

As the custom dialog class must extend the class `javax.swing.AbstractAction`, you can use the methods `getKeys`, `getValue` and `putValue` to find and set the value of the properties available in the dialog you are overwriting. `getKeys` will return an array of objects which have been set for the custom dialog you are overwriting. The objects may be an array themselves.

When overwriting an existing dialog, `getKeys` will contain the following objects:

`ACTIONID` : name of the action calling the dialog (see chapter [Actions](#)). `ACTIONID` is of type `java.lang.String` .

`SmallIcon` : the icon associated to the action. `SmallIcon` is of type `java.lang.String` .

`enabled` : determines if the action is enabled. `enabled` is of type `boolean` .

`cancelaction` : determines if the default action of the dialog should be executed. If set to `true`, the default action is not executed and only the custom dialog code is executed. If set to `false`, the default action is executed.

`defaultshortcut` : the default keyboard shortcut associated to the action. `defaultshortcut` is of type `java.lang.String` .

`apiobject` : a reference to the instance of the edit-on Pro applet (of type `com.realobjects.eop.japplet.SwingEditorApplet`)

`frame` : when constructing your dialogs, use this as parent. `frame` is of type `java.awt.Frame` .

`EOPDATA` : a hashtable containing key / value pairs providing more sophisticated possibilities to interact with the action. `EOPDATA` is of type `java.util.Hashtable` .

`defaultaction` : the default action object of the dialog as an `AbstractAction`. Can be used to call the default action within a customized standard action.

2.10.2.2 Interacting with `EOPDATA`

Depending on the action you are overwriting, the hashtable `EOPDATA` will contain another set of keys. For example, when you overwrite the `LINKPROPERTIES` action, `EOPDATA` will contain the following keys:

`allowedAttributes` : a list of the allowed attributes for the current element.

`attributes` : a hashtable containing name / value pairs of the attributes the element currently has

For a complete reference of the keys available when overwriting existing actions, please see the chapter [Keys available in EOPDATA for each action](#) .

2.10.2.3 Interacting with Custom Dialogs Derived from Custom Actions

When you do create a custom dialog by creating a custom action you are not able to interface with the applet using the `EOPDATA` hashtable. Instead, you can use the API methods described in the chapter [JavaScript API / Java API](#) .

Example: Using Java API methods in a custom dialog

```
SwingEditorApplet applet =
    (SwingEditorApplet)getValue("apiobject");
applet.insertHTMLData("<p>some content</p>");
```

2.10.3 Deploying a Custom Dialog

To deploy a custom dialog, you have to overwrite the corresponding action or create a new custom action in the `uiconfig.xml` file. For example, in case you want to overwrite the "Insert Hyperlink" dialog, you need to overwrite the `LINK` action. To do so, add the following line in the `<actions>...</actions>` section of the `uiconfig.xml`:

Example: Overwriting the `LINK` action

```
<actions>
  <action id="LINK"
    class="com.realobjects.customdialog.insertlink.InsertLinkDialog" />
</actions>
```

Use the class attribute to indicate the name of your class containing the custom dialog class. The class file must be located in the applet codebase. It can also be provided in a JAR file relative to the applet codebase. Use the JavaScript API method `addArchive` to load additional JAR archives in addition to applet archive. For details see the chapter [JavaScript API / Java API](#).

2.11 Using VersioTrack

VersioTrack is a powerful add-on for edit-on Pro which enables you to compare different versions of a document directly within edit-on Pro. It highlights and tracks all changes to the document, and allows you to reject or accept all or individual changes to a document in order to produce a final version of a document.

You will find information on how to compare two documents and reject / accept the changes within the document in this chapter.

2.11.1 Comparing Documents

There are multiple to load two documents in comparison mode. You can use the "Compare Documents" dialog to load the documents you want to compare, or one of the API functions `compareDocuments`, `compareDocumentContents`, `compareDocumentToEditorContent` and `compareDocumentContentToEditorContent`.

`compareDocuments` compares two documents from the specified URI to one another:

```
eop.compareDocuments (
    "http://www.myserver.com/doc1.html",
    "http://www.myserver.com/doc2.html"
);
eop.pumpEvents();
```

`compareDocumentContents` compares two documents passed as string to one another:

```
strDoc1 = "<p>Some content.</p>";
strDoc2 = "<p>Some more content.</p>";
eop.compareDocumentContents(strDoc1, strDoc2);
eop.pumpEvents();
```

`compareDocumentToEditorContent` compares a document from the specified URI to the editor content:

```
eop.compareDocumentToEditorContent ("http://www.myserver.com/doc1.html");
eop.pumpEvents();
```

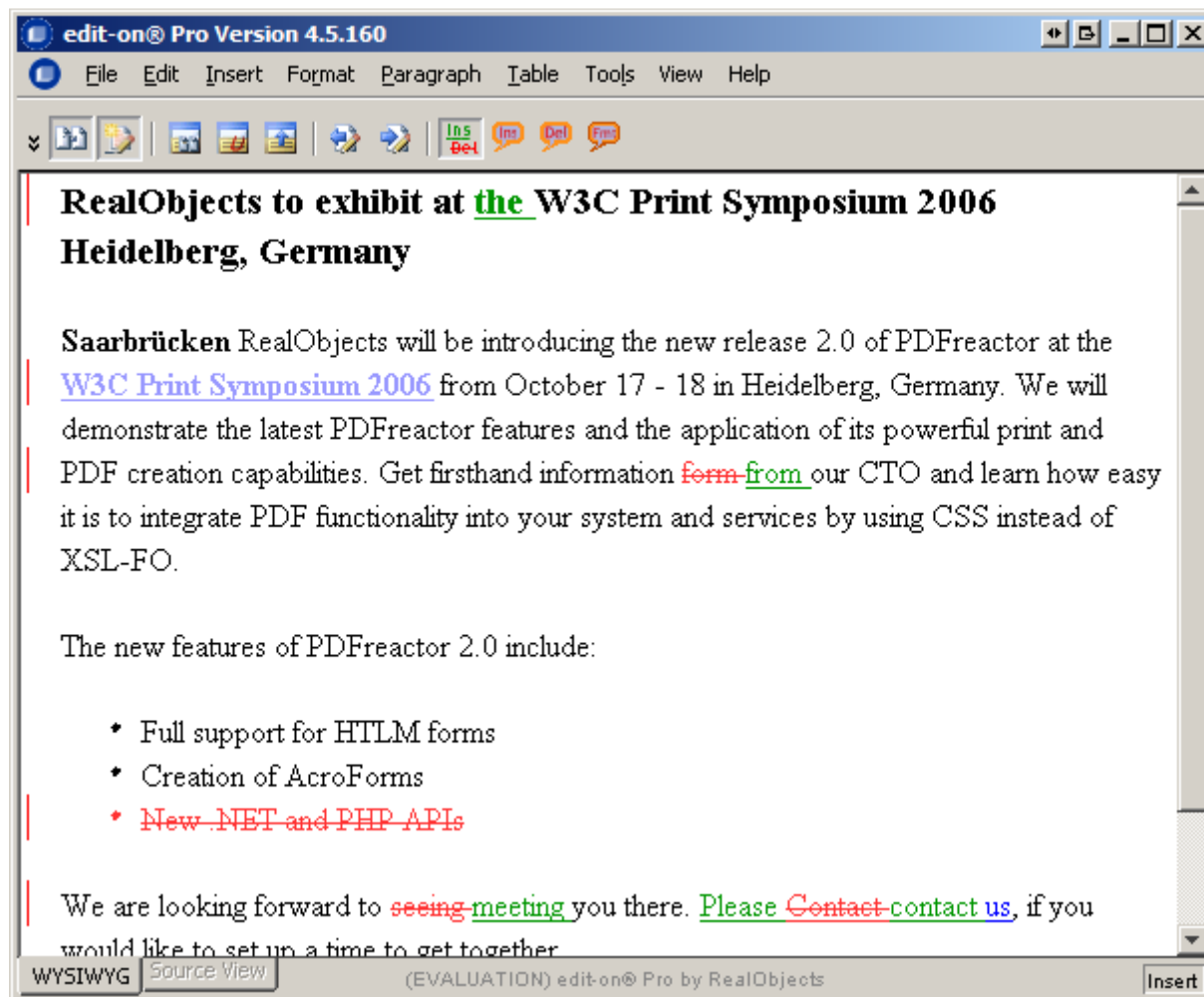
`compareDocumentContentToEditorContent` compares a document passed as string to the editor content:

```
strDoc = "<p>Some content.</p>";
eop.compareDocumentContentToEditorContent (strDoc);
eop.pumpEvents();
```

As an alternative to the API functions above, you can use the "Compare Documents" dialog to load the documents to be compared. You can integrate this dialog into your user interface using the action `DIFF`. For details please refer to the chapter [User Interface Configuration](#).

2.11.2 Comparison Mode

Once you have loaded the documents to be compared, edit-on Pro with VersioTrack will enter comparison mode as displayed below:



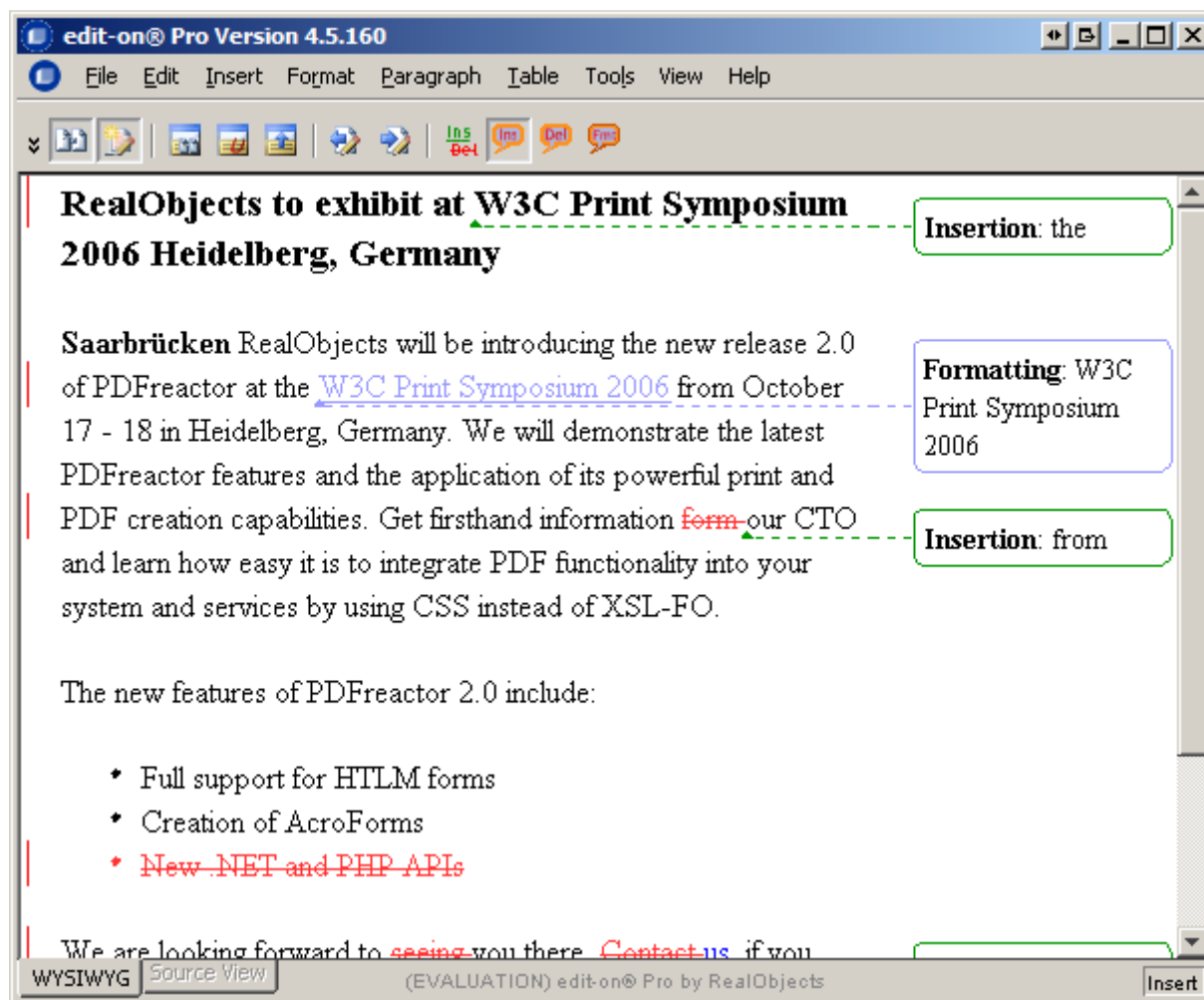
As you can see in the image above, new insertions are highlighted in green and underlined, while deletions are shown in red color and strike-through. Furthermore, formatting changes are highlighted in blue color and underlined.

Note that deletions are also indicated by a red vertical bar in the left margin of the document.

Now that the editor is in comparison mode, you have some tools at your disposition at your disposition which are not available in normal editing mode. You can accept or reject individual changes or all changes either directly, or using a dialog designed for this purpose. Furthermore you can decide to display the changes made to the original document, the final document, or formatting changes.

To display the original document, click "Show Original Document" in comparison mode. Example:

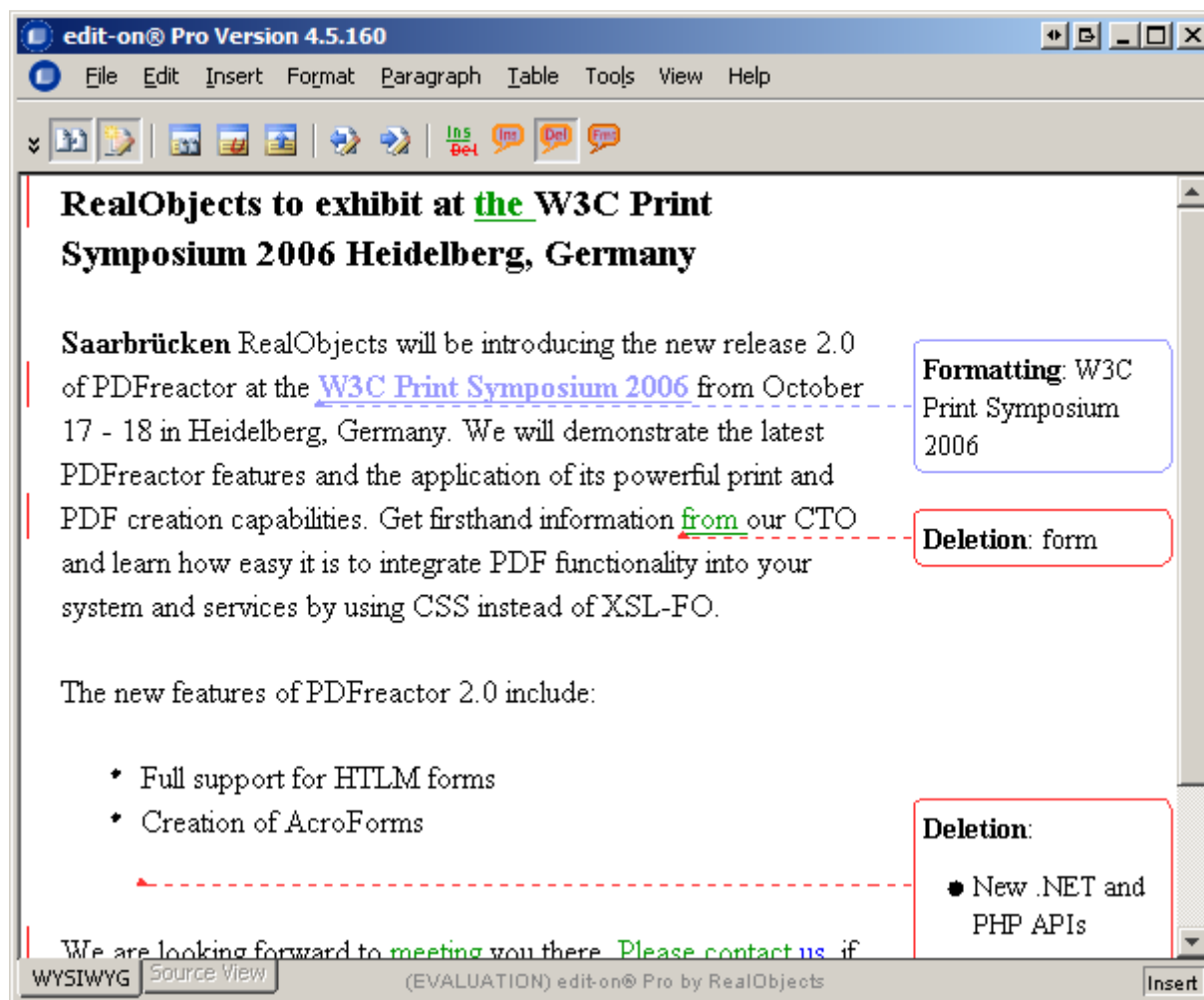
Original document view in comparison mode



In the original document view, all insertions made in the new document when compared to the old document are also shown as balloons on the right side of the editor canvas.

To display the final document, click "Show Final Document". Example:

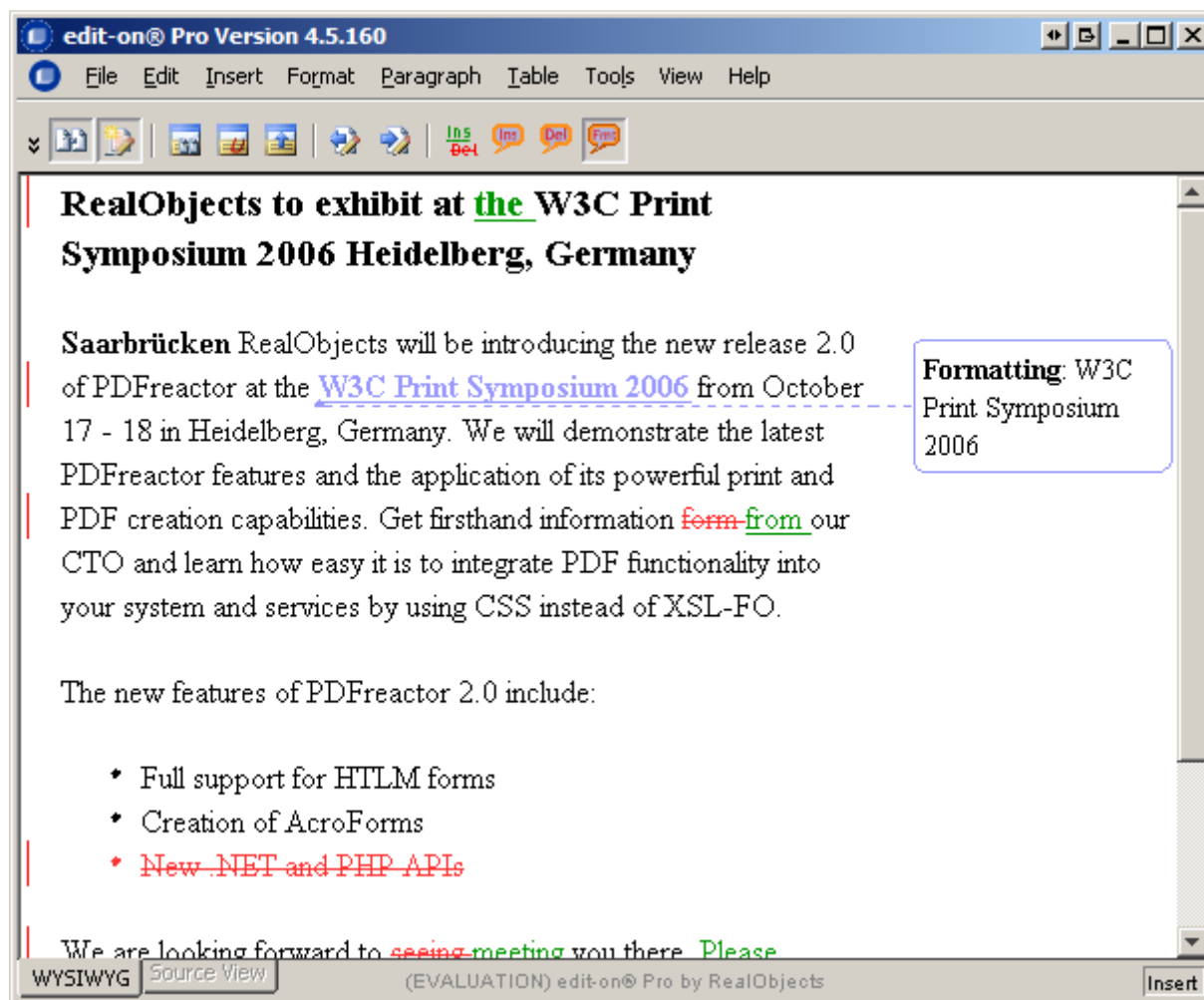
Final document view in comparison mode



In the final document view, all deletions and formatting changes which were made in the new document in contrast to the old document are also displayed in balloons on the right side of the editor canvas.

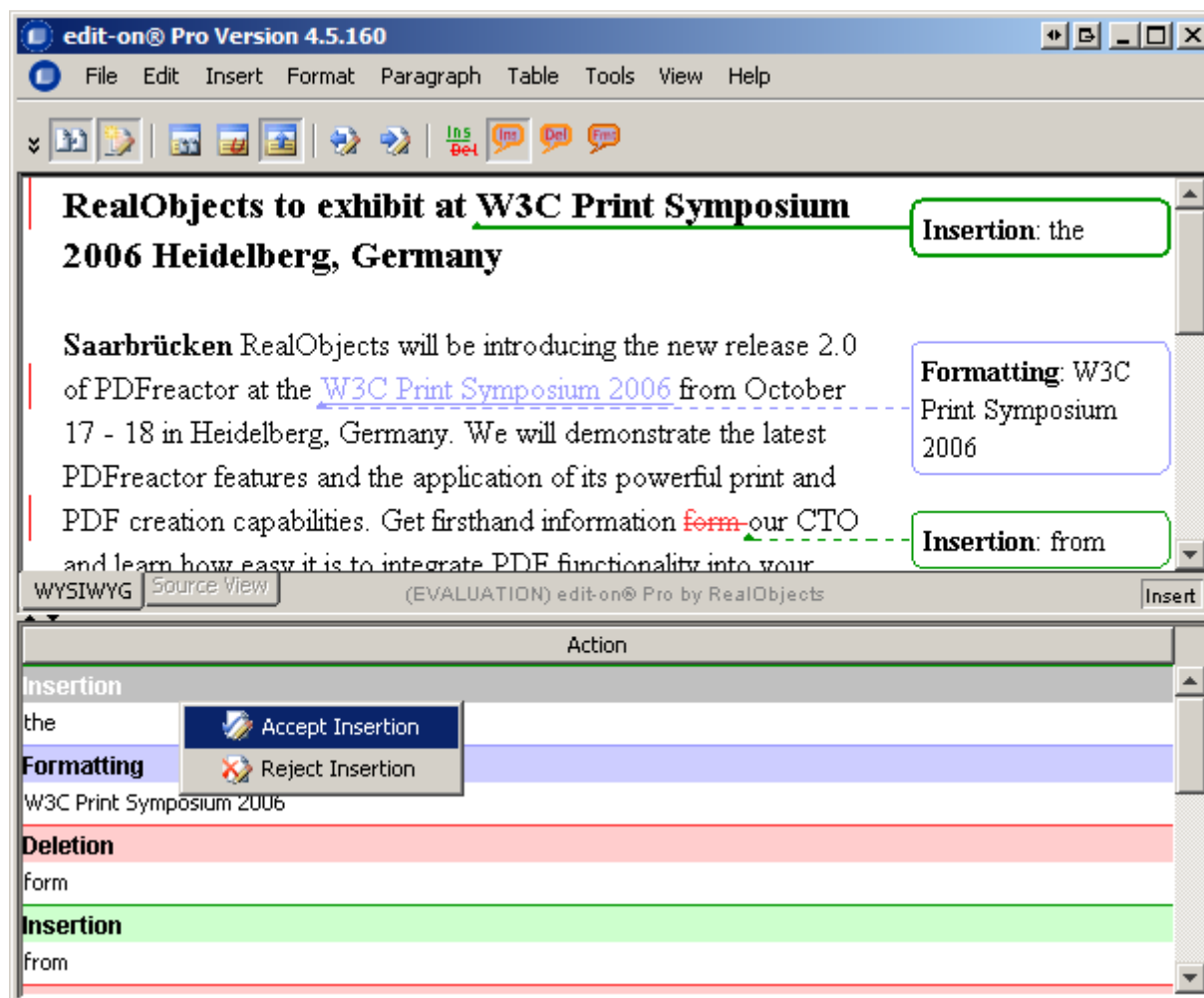
To display formatting changes, click "Show Formatting Changes". Example:

Formatting changes view



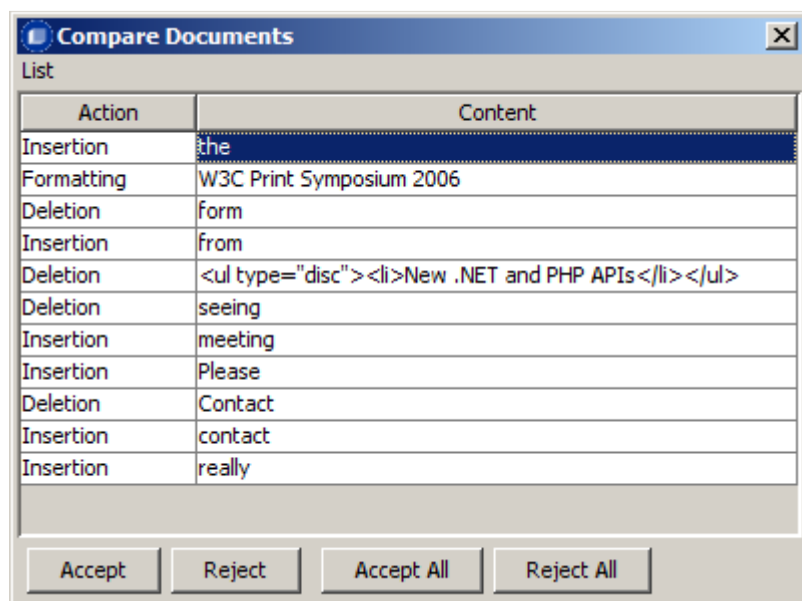
In formatting changes view, all formatting changes made in the new document are also displayed in balloons on the right side of the document canvas.

The changes made on the document can also be rendered in a list of changes instead of or in addition to the document itself. For example, clicking the "Diff Pane" button will display a list of changes made in the new document:



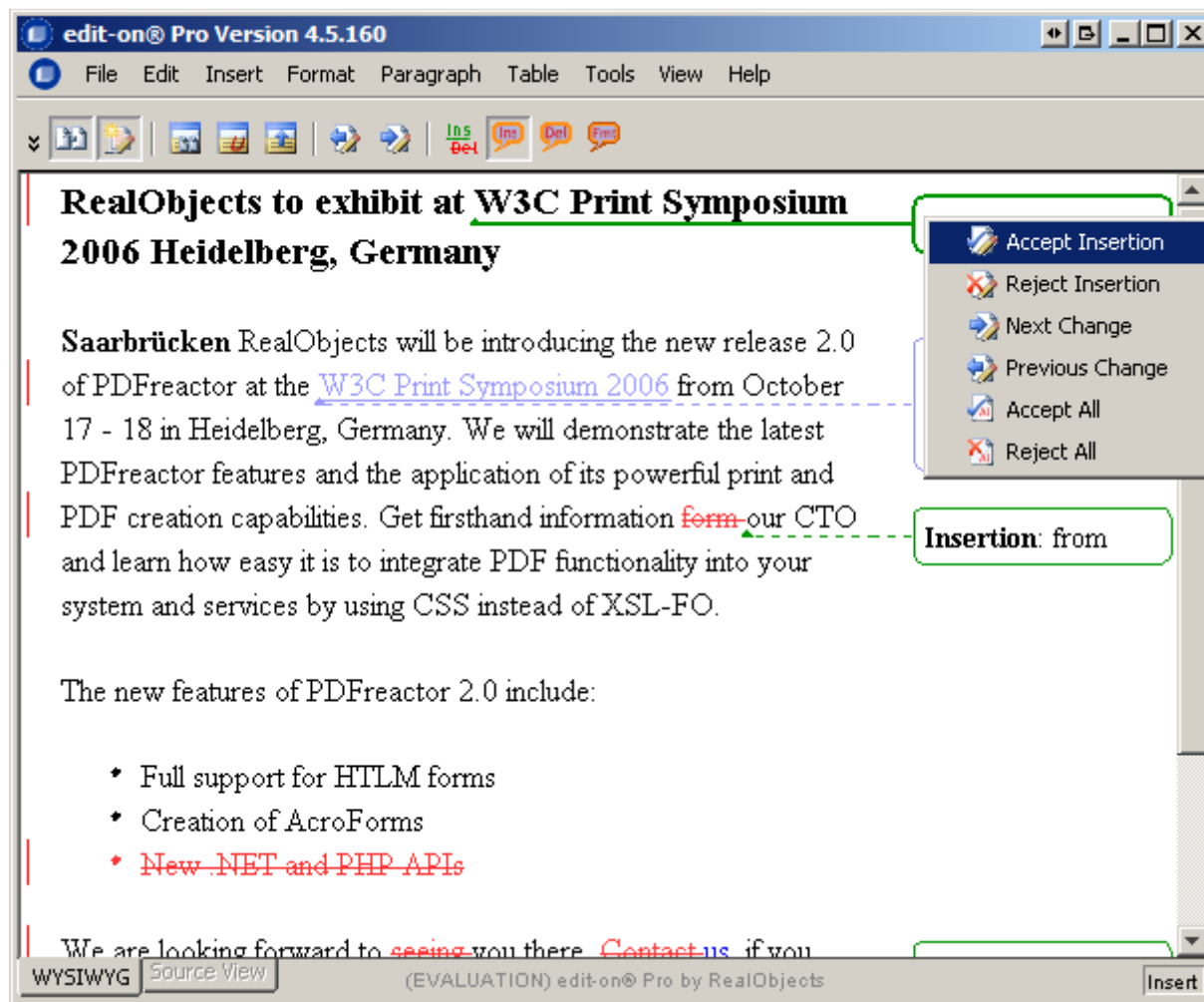
As displayed above, if one of the changes in the diff pane is clicked, it is also highlighted in darker color in the document itself. Furthermore, you can accept or reject changes directly from within the diff pane itself.

In addition to the diff pane, you can display all changes list format using the "Diff List" dialog:



As you can see above, the "Diff List" dialog also allows you to reject or accept individual or all changes made to the document.

You can of course also reject and accept changes directly within the document itself:



You can use the `PREVDIFF` and `NEXTDIFF` actions to browse through the changes made in the document.

Finally, once you have accepted and rejected all changes made within the document, you can leave comparison mode by calling the "Compare Documents" dialog again and confirming the changes made to the final document.

2.11.3 Comparison Mode 101

You will find some advice on using some of the comparison mode features and APIs here.

Disabled API Functions

First of all, most JavaScript API functions modifying the content of the editor are disabled in comparison mode. The following API functions are still available in comparison mode, but are subject to a different behaviour from their normal mode of operation:

```
setHTMLDataFromURL
setHTMLData
clear
compareDocuments
compareDocumentContents
```

`setHTMLDataFromURL` will end comparison mode, then load data from the specified URI.

`setHTMLData` will end comparison mode, then load data from the specified string.

`clear` will end comparison mode, then clear the editor contents.

`compareDocuments` will end comparison mode, load the documents to compare and enter comparison mode.

`compareDocumentContents` will end comparison mode, load the documents to compare and enter comparison mode.

Comparison Mode Event Handlers

As the behaviour of the editor significantly differs from its normal behaviour when in editing mode, you can use new event handlers to determine when comparison mode is entered or ended.

These event handlers are all set using the method `setEventHandler`. The available events are:

```
ONACCEPTINSERTIONFINISHED
ONREJECTINSERTIONFINISHED
ONACCEPTDELETIONFINISHED
ONREJECTDELETIONFINISHED
ONACCEPTFORMATFINISHED
ONREJECTFORMATFINISHED
ONACCEPTALLFINISHED
ONREJECTALLFINISHED
ONDIFFFINISHED
ONCANCELDIFFFINISHED
ONTRACKCHANGES
ONACCEPTSELECTEDCHANGESFINISHED
ONREJECTSELECTEDCHANGESFINISHED
```

Each of these events will be triggered under specific circumstances. For details please see [Event Handlers](#)

You can tie each of these events to a custom JavaScript function. The function will be called every time the event it is tied to is fired. Example:

```
function onDiffFinished() {
    data = eop.getHTMLData();
}

function onDiffCancelled() {
    eop.compareDocumentContents(doc1, doc2);
}

eop.setEventHandler("ONDIFFFINISHED", "onDiffFinished");
eop.setEventHandler("ONDIFFCANCELLED", "onDiffCancelled");
```

In the example above, the function `onDiffFinished` is called every time the user successfully ends comparison mode. Each time comparison mode is cancelled, the function `onCancelDiffFinished` is called.

2.11.4 Ignoring Changes

You can configure VersioTrack to ignore certain types of changes when in comparison mode. To do so, you have to set up an ignore filter.

The filter is defined in using XSLT and can be set using the JavaScript API method `setIgnoreFilter`. Here is a sample ignore filter:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">

  <xsl:template match="/">
    <xsl:copy>
      <xsl:apply-templates select="@*" />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*" />
    </xsl:copy>
  </xsl:template>

  <!-- sets the ignored element-->
  <xsl:template match="xhtml:p|xhtml:table">
    <xsl:copy>
      <xsl:attribute name="ignore-element">true</xsl:attribute>
      <xsl:apply-templates select="@*" />
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

For more information, please see <http://deltaxml.com/library/how-to-ignore-elements.html>

To set the filter we just defined as string, use `setIgnoreFilter`:

```
var filter = ... // The filter definition as XSLT expression, see above.
eop.setIgnoreFilter(filter);
eop.pumpEvents();
```

You can also load an ignore filter as file using a setting within the configuration file:

```
<diff>
  <ignorefilterurl>http://www.myserver.com/ignorefilter.xsl</ignorefilterurl>
</diff>
```

2.12 Event Handlers

edit-on Pro 5 introduces a new, simplified method to set event handlers. You can now set event handlers using simply the name of the event and the name function which should be called when this event is fired. Example:

```
function onDataPosted() {
  alert("Successfully posted data.");
}
eop.setEventHandler("ONDATAPOSTED", "onDataPosted");
eop.pumpEvents();
```

In this example, the function `onDataPosted` is called every time data is posted successfully.

The available events are:

```
ONACCEPTINSERTIONFINISHED
ONREJECTINSERTIONFINISHED
ONACCEPTDELETIONFINISHED
ONREJECTDELETIONFINISHED
ONACCEPTFORMATFINISHED
ONREJECTFORMATFINISHED
ONACCEPTALLFINISHED
ONREJECTALLFINISHED
ONDIFFFINISHED
ONCANCELDIFFFINISHED
ONTRACKCHANGES
ONACCEPTSELECTEDCHANGESFINISHED
ONREJECTSELECTEDCHANGESFINISHED
ONDATAPOSTEDERROR
ONEDITORFOCUSED
ONEDITORLOADED
ONDATALOADED
ONDATAPOSTED
ONEDITORFOCUSED
ONCHANGE
ONSELECTIONCHANGE
ONHTMLDATAINSERTED
ONSETTEMPLATE
```

ONACCEPTINSERTIONFINISHED: This event is thrown every time an insertion is accepted in comparison mode.

ONREJECTINSERTIONFINISHED: This event is thrown every time an insertion is rejected in comparison mode.

ONACCEPTDELETIONFINISHED: This event is thrown every time a deletion is accepted in comparison mode.

ONREJECTDELETIONFINISHED: This event is thrown every time a deletion is rejected in comparison mode.

ONACCEPTFORMATFINISHED: This event is thrown every time a formatting change is accepted in comparison mode.

ONREJECTFORMATFINISHED: This event is thrown every time a formatting change is rejected in comparison mode.

ONACCEPTALLFINISHED: This event is thrown when all changes are accepted in comparison mode.

ONREJECTALLFINISHED: This event is thrown when all changes are rejected in comparison mode.

ONDIFFFINISHED: This event is thrown when the comparison mode is successfully ended.

ONCANCELDIFFFINISHED: This event is thrown when cancelling the comparison mode.

ONTRACKCHANGES: This event is thrown when entering track changes mode.

ONACCEPTSELECTEDCHANGESFINISHED: This event is thrown when the changes within the current selection are accepted.

ONREJECTSELECTEDCHANGESFINISHED: This event is thrown when the changes within the current selection are rejected.

ONDATAPOSTEDERROR: This event is thrown when an error occurs while posting data using the direct HTTP Post API.

ONEDITORLOADED: This event is thrown when the editor was loaded.

ONDATALOADED: This event is thrown when data is loaded into the editor.

ONDATAPOSTED: This event is thrown when data was successfully posted using the direct HTTP Post API.

ONEDITORFOCUSED: This event is thrown when the editor receives the focus.

ONSELECTIONCHANGE: This event is thrown when the selection in the editor changes.

ONCHANGE: This event is thrown when the editor's content is modified.

ONHTMLDATAINSERTED: This event is thrown when content is inserted into the editor via the `insetHTMLData` or `insertHTMLDataFromURL` methods.

ONSETTEMPLATE: This event is thrown when a template is loaded by the editor. The template can either be specified in the configuration file or via the API methods `setTemplate` or `setTemplateURL`.

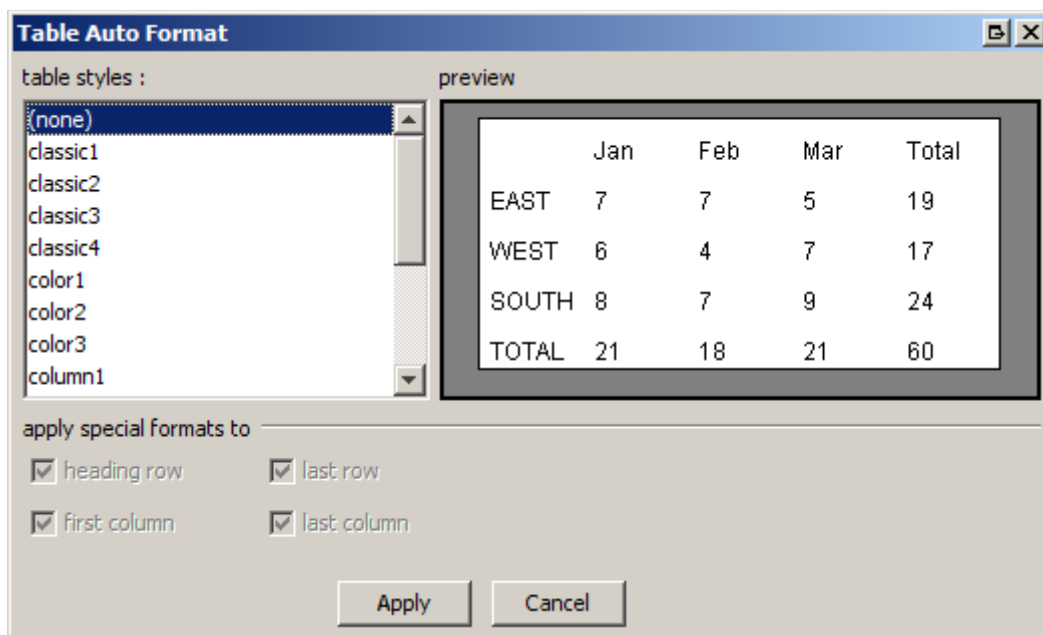
2.13 Configuring the Table Autoformat Feature

Using the table autoformat feature, you can insert a table with a predefined layout or apply a predefined format to an existing table.

When inserting a new table using the "Insert Table" dialog, you can select a table template by pressing the "AutoFormat..." button at the bottom of the dialog. This in turn opens the "Table Auto Format" dialog, which lets you select a table template to insert.

Alternatively, you can add the `TABLEAUTOFORMAT` action to your user interface and call this action when inside a table. This will let you select and apply a template to the current table. For details about the user interface configuration, see [Adjusting the User Interface](#).

A number of default templates are included with the editor by default. These templates are all pre-configured in the sample `config.xml` included within the edit-on Pro installation package.



If you do not want one or any of these templates to be available in the dialog, simply remove them from the configuration.

Example configuration:

```
<tableautoformat>
  <!-- default templates -->
  <tableautoformattemplateurl url="templates/classic1.xml" />
  <tableautoformattemplateurl url="templates/classic2.xml" />
  <tableautoformattemplateurl url="templates/classic3.xml" />
  <tableautoformattemplateurl url="templates/classic4.xml" />
  <tableautoformattemplateurl url="templates/color1.xml" />
  <tableautoformattemplateurl url="templates/color2.xml" />
  <tableautoformattemplateurl url="templates/color3.xml" />
  <tableautoformattemplateurl url="templates/column1.xml" />
  <tableautoformattemplateurl url="templates/column2.xml" />
  <tableautoformattemplateurl url="templates/column3.xml" />
  <tableautoformattemplateurl url="templates/contemp.xml" />
  <tableautoformattemplateurl url="templates/normal.xml" />
  <tableautoformattemplateurl url="templates/simple1.xml" />
  <tableautoformattemplateurl url="templates/simple2.xml" />
  <tableautoformattemplateurl url="templates/simple3.xml" />
</tableautoformat>
```

The default templates are all located within the edit-on Pro JAR file. You can of course specify a custom template by adding a `tableautoformattemplateurl` entry to the list of table templates and specifying the URL of the template. The URL may be relative to the editor code base.

Example configuration without any default templates and a single custom template:

```
<tableautoformat>
  <tableautoformattemplateurl url="http://server/template.xml" />
</tableautoformat>
```

The template itself is an XML file which specifies the structure of the table template. Example template file:

```
<tableautoformattemplate>
  <template name="simple1">
    <wholetable>
      <attribute name="cellspacing" value="0"/>
      <attribute name="cellpadding" value="0"/>
      <attribute name="border" value="1"/>
    </wholetable>
    <topleftcell>
      <attribute name="style"
        value="border:0.75pt solid green;"/>
    </topleftcell>
    <leftcolumn>
      <attribute name="style"
        value="border-style:none;"/>
    </leftcolumn>
    <firstrow>
      <attribute name="style"
        value="border:0.75pt solid green;"/>
    </firstrow>
    <rightcolumn>
      <attribute name="style"
        value="border-style:none;"/>
    </rightcolumn>
    <lastrow>
      <attribute name="style"
        value="border:0.75pt solid green;"/>
    </lastrow>
    <evenrowstripes>
      <attribute name="style"
        value="border-style:none;"/>
    </evenrowstripes>
    <bottomleftcell>
      <attribute name="style"
        value="border:none;"/>
    </bottomleftcell>
    <toprightcell>
      <attribute name="style"
        value="border-bottom:0.75pt solid green;"/>
    </toprightcell>
    <bottomrightcell>
      <attribute name="style"
        value=""/>
    </bottomrightcell>
    <oddrowstripes>
      <attribute name="style"
        value="border-style:none;"/>
    </oddrowstripes>
  </template>
</tableautoformattemplate>
```

In addition to being defined in separate files, table templates may also be defined directly within the config.xml. Example table autoformat configuration with all default templates being enabled, a custom template loaded from an external URI and a custom template defined directly:

```
<tableautoformat>
  <!-- default templates -->
  <tableautoformattemplateurl url="templates/classic1.xml" />
  <tableautoformattemplateurl url="templates/classic2.xml" />
  <tableautoformattemplateurl url="templates/classic3.xml" />
  <tableautoformattemplateurl url="templates/classic4.xml" />
```

```

<tableautoformattemplateurl url="templates/color1.xml" />
<tableautoformattemplateurl url="templates/color2.xml" />
<tableautoformattemplateurl url="templates/color3.xml" />
<tableautoformattemplateurl url="templates/column1.xml" />
<tableautoformattemplateurl url="templates/column2.xml" />
<tableautoformattemplateurl url="templates/column3.xml" />
<tableautoformattemplateurl url="templates/contemp.xml" />
<tableautoformattemplateurl url="templates/normal.xml" />
<tableautoformattemplateurl url="templates/simple1.xml" />
<tableautoformattemplateurl url="templates/simple2.xml" />
<tableautoformattemplateurl url="templates/simple3.xml" />
<tableautoformattemplateurl url="http://server/tabtemplate.xml"/>
<!-- custom templates -->
<tableautoformattemplateurl url="http://server/template.xml" />
<!-- custom template directly defined in the config -->
<tableautoformattemplate >
  <template name="direct">
    <wholetable>
      <attribute name="cellspacing" value="0"/>
      <attribute name="cellpadding" value="0"/>
      <attribute name="border" value="6"/>
    </wholetable>
    <toleftcell>
      <attribute name="style"
        value="border:0.75pt solid black;"/>
    </toleftcell>
    <leftcolumn>
      <attribute name="style"
        value="border:none;background-color:white;"/>
    </leftcolumn>
    <firstrow>
      <attribute name="style"
        value="border:0.75pt solid black;"/>
    </firstrow>
    <rightcolumn>
      <attribute name="style"
        value="border-style:none;color:black;background-color:white;"/>
    </rightcolumn>
    <lastrow>
      <attribute name="style"
        value="border:0.75pt solid black;"/>
    </lastrow>
    <evenrowstrips>
      <attribute name="style"
        value="border-style:none;color:black;background-color:white;"/>
    </evenrowstrips>
    <bottomleftcell>
      <attribute name="style"
        value="border:0.75pt solid black;font-weight:bold;"/>
    </bottomleftcell>
    <toprightcell>
      <attribute name="style"
        value="border:none;font-weight:bold;"/>
    </toprightcell>
    <bottomrightcell>
      <attribute name="style"
        value=""/>
    </bottomrightcell>
    <oddrowstrips>
      <attribute name="style"
        value="color:black;border-style:none;background-color:white;"/>
    </oddrowstrips>
  </template>
</tableautoformattemplate>
</tableautoformat>

```

2.14 Preloading the Applet and JVM

You can speed up the starting time of the editor by preloading the editor and or Java run-time environment.

On each start of the editor, the speed with which the editor is loaded mainly depends on two factors: whether the Java virtual machine is already started, and whether the applet JAR file is already cached in the JVM cache on the client side.

Usually at the very first start of the editor, neither of these factors will be the case, on thus the first start of the editor may take significantly longer than subsequent reloads of the editor.

Thus, you can now start the JVM and preload the editor JAR file into the JVM cache in the background of your page, independently of the editor itself. As this happens in the background, it does not affect your users and does not require starting the editor or integrating it into a page.

To this effect, the edit-on Pro JavaScript library contains two JavaScript functions, which allow you either to simply start the JVM before loading the applet, or to start the JVM and preload the applet JAR file into the JVM cache. These methods are `preloadJVM` and `preloadApplet` respectively.

It is highly recommended to include one of these functions in a page of your web application before the editor is actually loaded. If you are using `preloadJVM`, the JVM will already be started when accessing the page with the editor, which can cost quite some time on slower systems. However, on the very first start of the editor, it will still need to be downloaded into the client cache.

Example:

```
<script type="text/javascript" language="javascript" src="../../eopro/editonpro.js">
</script>

preloadApplet("eopro",
    true,
    "http://java.sun.com/products/plugin/autodl/jinstall-1_4-windows-i586.cab",
    "1,4,2");
```

It is highly recommended to preload the applet on a separate page that does not contain the editor itself, e.g. on the login screen of your application.

Important

Using the preload mechanism may cause a warning dialog to be displayed if Java 1.7.0_11 or newer is used. From this Java version on, this dialog is displayed for all unsigned applets.

3. Reference

Important:

All relative URLs used in configuration settings or JavaScript APIs are expanded with the codebase set for edit-on Pro. Setting the codebase is described in section [JavaScript API](#) .

3.1 Configuration File

config

Description:

Wrapper for the elements which specify the configuration of edit-on Pro.

Parent:

None.

Children:

fontmapping, presetcolors, customtags, spellingcheckers, cleanupprocess, defaulttablesettings, paragraphstyles, alternateenterkeyaction, fontsizeunit, fontsizes, documentcounter, tabpaneplacement, documentvalidation, sourceview, exportasnumerical, bodyonly, nbspfill, locale, uicolors, newdocumentdialog, multipleundoredo, oldfontstylemode, combowidth, standalonemode, licensekey, encoding, showall, sethtmldataurl, gethtmldataurl, eoeventhandling, ondataloaded, ondataposted, ondatapostederror, templateurl, customfield, localeurl, uicolor, dragndrop, directhttp, uploadimagesmethod, imagebase, presetsymbols

Parent:

None.

Example:

```
<config>
  <fontmapping usesystemfonts="true"/>
  <!-- further configuration settings ... -->
</config>
```

alternateenterkeyaction

Description:

Specifies the type of line break created when the enter key is pressed.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to false, a paragraph (`<p>...</p>`) is inserted when the enter key is pressed inside the editor. If `shift+enter` is pressed, a line break is created (`<br />`).

If `enabled` is set to true, a line break will be created when the enter key is pressed while a new paragraph is inserted when pressing `shift+enter`.

Example:

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

advanceddialog

Description:

Enables or disables the "Advanced" tab in all dialogs which contain an "Advanced" tab (e.g. the "Insert Image", "Insert Link", "Insert Hyperlink" dialogs). The "Advanced" tab of these dialogs lets the user specify values for attributes which can not be selected on the default tab of the dialog. The attributes that are available and the values that can be set are read directly from the DTD.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If `enabled` is set to false, the "Advanced" tab is not available.

If `enabled` is set to true, the "Advanced" tab can be used.

Example:

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

autohyperlink

Description:

Specifies if hyperlinking as-you-type is enabled. If this feature is enabled, when the user types in a hyperlink in WYSIWYG view, it will automatically be converted into a HTML hyperlink.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If `enabled` is set to true, hyperlinks typed in WYSIWYG View are automatically converted to HTML hyperlinks..

Example:

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

autosave**Description:**

Determines whether the editor's content should be regularly be saved. Also specifies in which interval the save should be performed, and which method should be used to save the editor's content.

Parent:

config

Children:

None.

Attributes:

`enabled (true|false) "true"`

If `enabled` is set to `false`, the autosave feature is disabled. If it is set to `true`, the content will be saved in an interval specified by the `interval` attribute.

`method (filesystem|post) "post"`

If `method` is set to `filesystem`, the editor's file will be saved to a file on the client computer. The file name and path is specified by the attribute `destination`.

If `method` is set to `post`, the editor's content is sent to the URL specified by `destination`. For details about how to handle POST requests sent by the editor, please see the chapter [Direct HTTP connection API](#).

`destination CDATA #REQUIRED`

The attribute `destination` specifies the location to which the editor's content will be saved to or posted to, depending on the selected autosave method.

`interval CDATA #REQUIRED`

The attribute `interval` specifies the interval after which the save request is repeated (in seconds).

Example:

```
<autosave enabled="true"
  method="filesystem"
  destination="c:\folder\myfile.htm"
  interval="300" />
```

base**Description:**

Specifies the URL used by the editor to find and display images or objects with a relative path. If this element is specified, the editor will look for images or objects with a relative path in a location relative to the `base`. If the element is not specified, it defaults to the code base.

Parent:

config

Children:

None,

Attributes:

`href`

Specifies the URL used by the editor to find and display images with a relative path.

Example:

```
<base href="http://myserver.com/" />
```

blockquote**Description:**

Specifies whether the "Insert Blockquote" dialog prompts the user to enter a value for the "cite" attribute of the "blockquote" element inserted by this dialog.

Parent:

config

Children:

None.

Attributes:

showinsertdialog (true|false) "true"

if set to true, "Insert Blockquote" dialog prompts the user to enter a value for the "cite" attribute of the "blockquote" element inserted by this dialog.

Example:

```
<blockquote showinsertdialog="false" />
```

bodyonly**Description:**

Specifies whether the editor exports content outside of the body (<body>...</body>) tag of a page.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to false, the editor will process content outside the page body and preserve header tags (e.g. <head>...</head>, <title>...</title>, etc...) . It will deliver a complete XHTML page when exporting.

If `enabled` is set to true, the editor will only process content inside the page body. The page body is all content inside of the <body>...</body> tags of a page. Always set `enabled` to true when dealing with fragments of web pages. This hashtable will contain status information about each file that was uploaded. It will contain the following info for each file: The old name of the file The new name of the uploaded file The new file path on the server The status code returned by the server The type of the upload that was performed (image, object or document)

Example:

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

browsefilesystem

Description:

Determines whether the "Browse File System" button is available in the "Insert Image" and "Insert Object" dialogs.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If `browsefilesystem` is set to true, the "Browse File System..." button will be available in the "Insert Image" and "Insert Object" dialogs.

If `browsefilesystem` is set to false, it will not be possible to select an image or object from the file system using the "Insert Image" and "Insert Object" dialogs.

Example:

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

cleanupprocess**Description:**

Specifies the (client-side) class or server-side script assuring the wellformedness of content imported into / exported from the editor. The clean-up will be assured by the editor itself by default. Use this attribute to override the clean-up process with your own implementation.

Parent:

config

Children:

None.

Attributes:

class CDATA #IMPLIED

The name of the class that used on the client-side to clean up data in the editor.

cleanuponload (off|auto|always) "auto"

Determines when a clean-up will be performed while content is loaded into the editor. If the attribute has the value `off`, the editor will never perform a clean-up on content that is imported.

If `cleanuponload` has been set to `auto`, edit-on Pro will perform a clean-up if the document that is imported is invalid in respect to the DocType specified in the editor.

If `cleanuponload` has been set to `always`, the clean-up will always be performed when content is imported.

cleanuponsave (off|auto|always) "auto"

Determines whether a clean-up will be performed while content is exported from the editor. If the attribute has the value `off`, the editor will not perform a clean-up while exporting content.

If `cleanuponsave` has the value `auto`, the editor will perform a clean-up if the document that is exported is invalid according to the DocType specified in the editor.

If `cleanuponsave` has the value `always`, the clean-up will always be performed when content is exported.

dropillegalattributes (true|false) "true"

Determines whether attributes not allowed in the Document Type Declaration used by the editor will be removed from the document.

dropillegaltags (true|false) "true"

Determines whether elements not allowed in the Document Type Declaration used by the editor will be removed from the document.

url CDATA #IMPLIED

Determines the URL of the server-side script cleaning the data in the editor.

hidecomments (true|false) "false"

If set to `true`, MS Word generated comments are removed when content is pasted from MS Word.

loadandignorecleanuperrors (true|false) "false"

If set to `true`, cleanup errors are ignored when trying to load a document. In this mode the editor tries to load the document no matter what and will not display a warning dialog if an invalid document or a document that is not wellformed is loaded. Use this feature at your own risk.

Example:

```
<cleanupprocess url="http://myserver.com/customCleanUp.jsp"
  dropillegalattributes="false" />
```

converttablewidthtorelative**Description:**

If this is set, any table with an absolute size pasted into the editor will be converted to a table with relative width.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If this is set, any table with an absolute size pasted into the editor will be converted to a table with relative width.

reference CDATA #IMPLIED

Determines a width (in pixels) relative to which tables with an absolute width will be converted to a relative width.

Example:

```
<converttablewidthtorelative enabled="true" reference="800" />
```

customfield**Description:**

May be used to specify an additional custom field sent to the server-side script specified by the element `directhttp` along with the field `HTMLDATA`. The request will be a POST request and the data will be stored in a field called `CUSTOMFIELD`.

Parent:

config

Children:

None.

Attributes:

value CDATA #REQUIRED

Determines the value that the field `CUSTOMFIELD` will contain.

Example:

```
<customfield value="CustomFieldValue" />
```

customtagseditingonly**Description:**

If this setting is enabled, editing will only be possible within custom tags in the document.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If this is set to true, editing will only be possible within custom tags in the document.

Example:

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

defaulttablesettings**Description:**

Specifies the default values of the default values that will be preset in the `Insert Table` dialog. A table inserted using the `INSERTTABLE` action will also have these values.

Parent:

`config`

Children:

`table_bgcolor`, `table_borderwidth`, `table_caption`, `table_cellpadding`, `table_cellspacing`, `table_colwidth`, `table_maxcols`, `table_maxrows`, `table_numcols`, `table_numrows`, `table_overallwidth`, `table_row`

Attributes:

`table_align` (`left` | `right` | `center`)

If set, tables inserted using the `Insert Table` dialog will have the specified alignment by default.

`cell_valign` (`top` | `middle` | `bottom` | `baseline`)

If set, the cells of tables inserted using the `Insert Table` dialog will have the specified vertical alignment by default.

`width_unit` (`percent` | `pixel`)

Allows to specify whether the table and cell width should be set in pixel or percent by default.

`height_unit` (`percent` | `pixel`)

Allows to specify whether the table and cell height should be set in pixel or percent by default.

`automatic_relative_cellwidth` (`true|false`) `"false"`

If this is set to `true`, when a table with a relative width is inserted and no width was specified for the individual cells, width attributes will automatically be added to the tables' cells. These width attributes will of course also have relative values and the editor will make sure the total width for all cells in a row is 100%.

`widthinstyle` (`true|false`) `"false"`

If this is set to `true`, the table width will be set using style properties instead of the width attribute.

Example:

```
<defaulttablesettings table_align="center"
  cell_valign="middle"
  width_unit="percent"
  height_unit="percent">
  <table_bgcolor>#dedfde</table_bgcolor>
  <table_borderwidth>2</table_borderwidth>
  <table_caption alignment="top">Test Caption</table_caption>
  <table_cellpadding>3</table_cellpadding>
  <table_cellspacing>2</table_cellspacing>
  <table_colwidth>20%</table_colwidth>
  <table_maxcols>10</table_maxcols>
  <table_maxrows>10</table_maxrows>
  <table_numcols>5</table_numcols>
  <table_numrows>5</table_numrows>
  <table_overallwidth>100%</table_overallwidth>
  <table_rowheight>10%</table_rowheight>
</defaulttablesettings>
```

table_bgcolor**Description:**

Specifies the default background color set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_bgcolor>#dedfde</table_bgcolor>
```

table_borderwidth**Description:**

Specifies the default table border width color set in the `Insert Table` dialog. The default is "1".

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None. If this is set, any table with an absolute size pasted into the editor will be converted to a table with relative width.

Example:

```
<table_borderwidth>25%</table_borderwidth>
```

table_caption**Description:**

Specifies the default table caption set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

`alignment (left|right|top|bottom) #IMPLIED`

Specifies the alignment of the table caption.

Example:

```
<table_caption alignment="top">
  Test caption
</table_caption>
```

table_cellpadding

Description:

Specifies the default cell padding set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_cellpadding>2</table_cellpadding>
```

table_cellspacing

Description:

Specifies the default cell spacing set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.isbottom

Example:

```
<table_cellspacing>2</table_cellspacing>
```

table_colwidth

Description:

Specifies the default column width color set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_colwidth>25%</table_colwidth>
```

table_maxcols**Description:**

Specifies the maximum number of table columns that can be inserted using the `Insert Table` dialog.

Parent:

`inserttabledialog`

Children:

None.

Attributes:

None.

Example:

```
<table_maxcols>15</table_maxcols>
```

Note:

Setting the number of max cols above 10 may affect the editor's performance on some systems

table_maxrows**Description:**

Specifies the maximum number of table rows that can be inserted using the `Insert Table` dialog.

Parent:

`inserttabledialog`

Children:

None.

Attributes:

None.

Example:

```
<table_maxrows>15</table_maxrows>
```

Note:

Setting the number of max rows above 10 may affect the editor's performance on some systems

table_numcols**Description:**

Specifies the number of columns set in the `Insert Table` dialog. Default value: 2.

Parent:base

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_numcols>4</table_numrows>
```

table_numrows

Description:

Specifies the number of rows set in the `Insert Table` dialog. Default value: 2.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_numrows>3</table_numrows>
```

table_overallwidth

Description:

Specifies the default width set in the `Insert Table` dialog. Default value: 100%.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_overallwidth>100%</table_overallwidth>
```

table_rowheight

Description:

Specifies the default row height set in the `Insert Table` dialog.

Parent:

`defaulttablesettings`

Children:

None.

Attributes:

None.

Example:

```
<table_rowheight>10</table_rowheight>
```

diff**Description:**

Allows the configuration of the display and ignore filter of the comparison mode.

Parent:

config

Children:

ignorefilterurl, verticalbarlook

Attributes:

None.

Example:

```
<diff>
  <verticalbarlook>
    verticalbar-width="2"
    verticalbar-color="#0689cc"
  />
  <ignorefilterurl>http://www.yourserver.com/ignorefilter.xml</ignorefilterurl>
</diff>
```

ignorefilterurl**Description:**

You can configure VersioTrack to ignore certain types of changes when in comparison mode. To do so, you have to set up an ignore filter. The ignore filter is in XSLT format. You can specify the location of the ignore filter as content of the `ignorefilterurl` element.

See [Ignoring Changes](#) for details.

Parent:

config

Children:

None.

Attributes:

None.

Example:

```
<ignorefilterurl>http://www.yourserver.com/ignorefilter.xml</ignorefilterurl>
```

verticalbarlook**Description:****Parent:**

diff

Children:

None.

Attributes:

verticalbar-width CDATA #IMPLIED

Specifies the width of the vertical bar that marks changes on the left side of the editor in comparison mode.

vertical-color CDATA #IMPLIED

Specifies the color of the vertical bar that marks changes on the left side of the editor in comparison mode.

Example:

```
<ignorefilterurl>http://www.yourserver.com/ignorefilter.xml</ignorefilterurl>
```

defaultfont

Description:

Specifies the default font used to display the editor's content where no font is otherwise specified.

This font is only used to render the editor's content, and is not exported in any way.

Parent:

config

Children:

None.

Attributes:

name CDATA #REQUIRED

Specifies the default font used to display the editor's content where no font is otherwise specified.

This font is only used to render the editor's content, and is not exported in any way.

Example:

```
<defaultfont name="Arial"></defaultfont>
```

directhttp**Description:**

Allows the configuration of the HTTP direct connection API.

Parent:

config

Children:

None.

Attributes:

sethtmldataurl CDATA #REQUIRED

Determines the location of a document loaded after the editor is initialized.

setokurl CDATA #IMPLIED

Allows to specify an URL called after a document specified by the attribute `url` has been loaded successfully.

setoktarget CDATA #IMPLIED

Determines the target frame of the document specified by the `setokurl` attribute.

seterrorurl CDATA #IMPLIED

Allows to specify a URL called if the document specified by the attribute `url` could not be successfully loaded.

seterrortarget CDATA #IMPLIED

Determines the target frame of the document specified by the `seterrorurl` attribute.

gethtmldataurl CDATA #REQUIRED

Specifies the location of the server-side script that will receive the data posted by the editor when the `SAVE` action is fired. The request will be a POST request and the data will be stored in a field with the name `HTMLDATA`.

getokurl CDATA #IMPLIED

Allows to specify an URL to be called after the editor's content has been sent successfully to the location specified by the attribute `url`.

getoktarget CDATA #IMPLIED

Determines the target frame of the document specified by the `getokurl` attribute.

geterrorurl CDATA #IMPLIED

Allows to specify an URL called if the editor's content couldn't be sent successfully to the location specified by the attribute `url`.

geterrortarget CDATA #IMPLIED

Determines the target frame of the document specified by the `geterrorurl` attribute.

Example:

```
<directhttp sethtmldataurl="http://myserver.com/mycontent.htm"
  setokurl="http://myserver.com/ok.htm"
  setoktarget="_new"
  seterrorurl="http://myserver.com/error.htm"
  seterrortarget="_blank"
  gethtmldataurl="http://myserver.com/save.php"
  getokurl="http://myserver.com/ok.htm"
  getoktarget="_new"
  geterrorurl="http://myserver.com/error.htm"
  geterrortarget="_blank" />
```

documentcounter**Description:**

Enables character counting and allows to restrict the number of characters in a document to a specified maximum.

Parent:

config

Children:

None.

Attributes:

limit CDATA #IMPLIED

Determines the number of characters displayed as upper limit in the editor.

enforcelimit (true|false) "false"

Determines whether the character limit specified by the attribute `limit` will be enforced or not. If set to true, adding further content will be blocked if the limit is surpassed.

Example:

```
<documentcounter limit="5000" enforcelimit="true" />
```

documentvalidation**Description:**

Specifies whether the editor's content will be validated against the DocType set in the editor.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If `enabled` is set to false, the editor will not validate documents.

If `enabled` is set to true, documents loaded into the editor will be validated against the DocType set in the editor.

validateonsave (true|false) "false"

If `validateonsave` is set to true, documents will be validated against the DTD specified by the DocType set in the editor when they are saved.

overridedoctype (true|false) "true"

If `overridedoctype` is set to false, the editor will always validate against the DocType of the document loaded into the editor.

If `overridedoctype` is set to true, the editor will always validate against the DocType specified in the template loaded in the editor and ignore the current document type set in the editor.

allowforcedload (true|false) "false"

If `allowforcedload` is set to true, a prompt will appear asking for confirmation when ever the user tries to load an invalid document. If the user selects "Load anyway" from the prompt, the editor will try to load the document anyway. If this attribute is set to false, the user will not be able to enforce the loading of invalid documents.

Example:

```
<documentvalidation enabled="true"
  validateonsave="true"
  overridedoctype="true" />
```

dragndrop

Description:

Specifies whether drag and drop is enabled in the editor or not.

Parent:

configbase

Children:

None.

Attributes:

enabled (true|false) "true"

If `enabled` is set to false, it is not possible to drag and drop external documents into the editor.

Example:

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

encoding

Description:

Specifies the encoding used by the editor.

Parent:

config

Children:

None.

Attributes:

value CDATA #REQUIRED

Specifies the encoding used by the editor. See <http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html> for possible values.

Note:

The name of a specific encoding used as value for this element may differ from the HTML term for this encoding. The name used by the `encoding` element is the Java term for this encoding and can be found in the document specified by the link above.

Important:

The web page containing the editor must use the same encoding as the editor in the document loaded in the editor. For example, in order to use the UTF-8 encoding, the value of the `encoding` element must be "UTF8", while "utf-8" has to be used in the page's META tag.

Example:

```
<encoding value="UTF8" />
```

eopeventhandling**Description:**

Wrapper for the editor's event handling elements, which allow to specify JavaScript functions executed when a specific event is fired.

Parent:

config

Children:

ondataloaded, ondataposted, ondatapostederror

Attributes:

None.

Example:

```
<eopeventhandling>
  <ondataloaded value="OnDataLoaded" />
  <ondataposted value="OnDataPosted" />
  <ondatapostederror value="OnDataPostedError" />
</eopeventhandling>
```

ondataloaded**Description:**

Specifies a JavaScript function to be executed after content was loaded into the editor using the `setHTMLDataFromURL()` function.

Parent:

eopeventhandling

Children:

None.

Attributes:

value CDATA #REQUIRED

Specifies a JavaScript function to be executed after content was loaded into the editor using the `setHTMLDataFromURL()` function. The specified JavaScript function will also receive a reference to the applet as first argument.

Example:

```
<ondataloaded value="OnDataLoaded" />
```

ondataposted**Description**

Specifies a JavaScript function to be executed after the editor's content was successfully sent to the location specified by the `gethtmldataurl` attribute of the element `directhttp`.

Parent:

eopeventhandlingbase

Children:

None.

Attributes:

value CDATA #REQUIRED

Specifies a JavaScript function to be executed after the editor's content was successfully sent to the location specified by the element `directhttp`. The specified JavaScript function will also receive a reference to the applet as first argument.

Example:

```
<ondataposted value="OnDataPosted" />
```

ondatapostederror**Description:**

Specifies a JavaScript function to be executed if the editor's content was not sent successfully to the location specified by the `gethtmldataurl` attribute of the element `directhttp` in the configuration file.

Parent:

`eoeventhandling`

Children:

None.

Attributes:

`value` CDATA #REQUIRED

Specifies a JavaScript function to be executed if the editor's content was not sent successfully to the location specified by the element `directhttp`. The specified JavaScript function will also receive a reference to the applet as first argument.

Example:

```
<ondatapostederror value="OnDataPostedError" />
```

exportasnumerical**Description:**

Specifies whether the editor should export special characters and symbols (as defined in XHTML entity sets) as numeric character entities.

Parent:

`config`

Children:

None.

Attributes:

`enabled` (true|false) "false"

If `enabled` is set to true, the editor will export special characters and symbols as numeric character entities. E.g. "ä" will be exported as "ä".

Example:

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

Note:

Characters will only be exported as entities if they are not supported by the encoding specified in the document. If characters are supported by the specified encoding, they will be exported as such.

fontmapping

Description:

Wrapper for the `font` element, specifying to which font the fonts in the editor will be mapped to when the document is exported.

Parent:

`config`

Children:

`font`

Attributes:

`usesystemfonts` (true|false) "false"

If `usesystemfonts` is set to true, all fonts available on the client system will also be available in the editor.

Example:

```
<fontmapping usesystemfonts="true" />
```

Note:

It is recommended to set `usesystemfonts` to true only in an homogeneous environment (i.e. where all client machines using the editor have a similar configuration), as fonts available on a specific client system may not be available on other client systems. Documents using a specific font may not be displayed correctly on all client systems viewing the document.

font

Description:

Specifies a mapping of a font within the editor.

Parent:

`fontmapping`

Children:

`renderedfont`, `exportedfont`

Attributes:

`name` CDATA #REQUIRED

Determines the name of the font displayed within the editor.

Example:

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

renderedfont**Description:**

Specifies the font used in the editor to render this font name.

Parent:

font

Children:

None.

Attributes:

url CDATA #IMPLIED

If a specific font is not available on all client machines, this font can be fetched from a font file in TTF format at the URL specified by `url`. This font is then used to render the font specified by `renderedfont`.

Example:

```
<fontmapping>
  <font name="Roman">
    <renderedfont
      url="http://myserver/customfont.ttf">Serif</renderedfont>
    <exportedfont>Georgia, Times New Roman, Times</exportedfont>
  </font>
</fontmapping>
```

exportedfont**Description:**

Specifies the name of the font this font name will be exported to.

Parent:

font

Children:

None.

Attributes:

None.

Example:

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

fontsizeunit**Description:**

Specifies the unit used to display fonts.

Parent:

config

Children:

None.

Attributes:

unit (pt|px|em|cm|%) "px"

Determines the font size unit used in the editor. The default unit is "px".

Example:

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

fontsizes**Description:**

Wrapper for the `fontsize` element, specifying the font sizes available in the font size selector box.

Parent:

`config`

Children:

`fontsize`

Attributes:

None.

Example:

```
<fontsizes>
  <fontsize size="12em" />
  <fontsize size="12cm" />
  <fontsize size="12%" />
  <fontsize size="large" />
  <fontsize size="24" />
  <fontsize size="36" />
</fontsizes>
```

fontsize**Description:**

Specifies a font size available in the font size selector box.

Parent:

`fontsizes`

Children:

None.

Attributes:

`size` CDATA #REQUIRED

Specifies a font size available in the font size selector box. The font size may be specified by a unit descriptor such as "cm", "%" or "em" to determine which font size unit is used. If no unit is specified, the size unit specified by the element `fontsizeunit` will apply.

Example:

```
<fontsize size="12" />
```

iconrepository

Description:

Specifies a repository from which the editor loads the toolbar icons. If this repository is specified, the applet will first look for icons in the repository. If the icon can not be found in the repository, it is loaded from the edit-on Pro JAR file.

If a relative path is used for the repository, it is resolved relative to the editor's codebase.

Parent:

config

Children:

None.

Attributes:

url

Specifies a repository from which the editor loads the toolbar icons. If this repository is specified, the applet will first look for icons in the repository. If the icon can not be found in the repository, it is loaded from the edit-on Pro JAR file.

Example:

```
<iconrepository url="myicons/" />
```

imagebase

Description:

Specifies the URL used by the editor to find and display images with a relative path. If this element is specified, the editor will look for images with a relative path in a location relative to the `imagebase`. If the element is not specified, it defaults to the code base.

Parent:

config

Children:

None.

Attributes:

url

Specifies the URL used by the editor to find and display images with a relative path.

Example:

```
<imagebase url="http://myserver.com/myimages" />
```

Important:

This setting is deprecated as of edit-on Pro version 4.2. Please use `base` instead.

inlinequote**Description:**

Specifies whether the "Insert Inline Quote" dialog prompts the user to enter a value for the "cite" attribute of the "inlinequote" element inserted by this dialog.

Parent:

config

Children:

None.

Attributes:

showinsertdialog (true|false) "true"

if set to true, "Insert Inline Quote" dialog prompts the user to enter a value for the "cite" attribute of the "q" element inserted by this dialog.

Example:

```
<blockquote showinsertdialog="false" />
```

licensekey**Description:**

Specifies the location of the license key file required by the editor.

Parent:

config

Children:

None.

Attributes:

value CDATA #IMPLIED

Specifies the URL location of the license key file. The value may be relative to the editor's codebase. If this element is not specified, the editor will search for a file called `licensekey.xml` at the editor's codebase.

Example:

```
<licensekey value="licensekey.xml" />
```

locale**Description:**

Specifies the language module defining the user interface language.

Parent:

config

Children:

None.

Attributes:

value CDATA #IMPLIED

Specifies the language module defining the user interface language. The possible values are American English ("en_US"), Spanish ("es_ES"), French ("fr_FR") and German ("de_DE").

Example:

```
<locale value="en_US" />
```

localeurl**Description:**

Specifies the location of the locale file used by the editor. The value may be relative to the applet codebase. If specified, it overrides the value set by `locale.localeurl` enables the integrator to specify a custom locale not available in the editor by default.

Parent:

config

Children:

None.

Attributes:

value CDATA #IMPLIED

Specifies the location of the locale file used by the editor.

Example:

```
<localeurl value="locale_it_IT.xml" />
```

mousewheelscroll**Description:**

Specifies the scroll increment per rotation of the mouse wheel. The default scroll increment is 172.

Parent:

config

Children:

None.

Attributes:

increment CDATA #REQUIRED

Specifies the scroll increment per rotation of the mouse wheel. The default scroll increment is 172.

Example:

```
<mousewheelscroll increment="400" />
```

multipleundoredo**Description:**

Specifies whether it is possible to undo / redo a sequence of actions performed within the editor.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to `false`, it is not possible to undo / redo a sequence of actions performed within the editor.

maxsteps CDATA #REQUIRED

Determines the maximum number of actions which can be undone / redone in sequence within the editor.

Example:

```
<multipleundoredo enabled="true" maxsteps="5" />
```

newdocumentdialog**Description:**

Specifies whether the editor will prompt for confirmation before clearing the content when the `NEW` action is fired.

Parent:

`config`

Children:

None.

Attributes:

`enabled (true|false) "true"`

If `enabled` is set to `true`, the editor will prompt for confirmation before clearing the content when the `NEW` action is fired.

Example:

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

nbspcfill**Description:**

Specifies whether a " " entity will be inserted into empty paragraphs when the editor's content is exported.

Parent:

`config`

Children:

None.

Attributes:

`enabled (true|false) "false"`

If `enabled` is set to `true`, " " entities will be inserted in empty paragraphs outside tables when the editor's content is exported.

`filltable (true|false) "false"`

If `filltable` is set to `true`, " " entities will be inserted in empty table cells when the editor's content is exported.

Example:

```
<nbspcfill enabled="true" filltable="true" />
```

oldfontstylemode**Description:**

Specifies whether the editor will use the "old font style" mode or not.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to true, the "", "<i>" and "<u>" tags will be used instead of "", "" and "" respectively when entering bold, italic or underlined text.

If `enabled` is set to false, "", "" and "" will be used for bold, italic and underlined text.

Example:

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

pageproperties**Description:**

Allows to configure whether certain tabs in the "Page Properties..." dialog should be disabled.

Parent:

config

Children:

None.

Attributes:

disablegeneraltab (true|false) "false"

If set to true, the "General" tab of the "Page Properties" dialog is disabled.

disablecolorstab (true|false) "false"

If set to true, the "Colors" tab of the "Page Properties" dialog is disabled.

disablebackgroundtab (true|false) "false"

If set to true, the "Background" tab of the "Page Properties" dialog is disabled.

Example:

```
<pageproperties disablecolorstab="true" />
```

paragraphstyles

Description:

Wrapper for the `paragraphstyle` element specifying the styles that can be set in the `Paragraph Styles` selector box.

Parent:

`config`

Children:

`paragraphstyle`

Attributes:

None.

Example:

```
<paragraphstyles>
  <paragraphstyle name="h1" />
  <paragraphstyle name="h2" />
  <paragraphstyle name="h3" />
  <paragraphstyle name="h4" />
</paragraphstyles>
```

paragraphstyle

Description:

Specifies the individual paragraph styles available in the `Paragraph Styles` selector box.

Parent:

`paragraphstyles`

Children:

None.

Attributes:

`name`

Specifies the individual paragraph styles available in the `Paragraph Styles` selector box.

Example:

```
<paragraphstyle name="h1" />
```

presetcolors

Description:

Wrapper for the `color` element specifying the preset colors available in the applet's color dialog.

Parent:

`config`

Children:

`color`

Attributes:

`only (true|false) "false"`

If `only` is set to `true`, only preset colors specified by the integrator will be available in the editor.

Example:

```
<presetcolors>
  <color rgb="#ff0000" desc="Red" />
  <color rgb="#ffffff" desc="White" />
  <color rgb="#000000" />
</presetcolors>
```

color**Description:**

Specifies a custom color available in the applet's color dialog.

Parent:

presetcolors

Children:

None.

Attributes:

rgb CDATA #REQUIRED

Specifies the RGB value of the color.

desc CDATA #IMPLIED p

Specifies a description of the color which will appear in the color dialog.

Example:

```
<color rgb="#ff0000" desc="Red" />
```

pastespecial**Description:**

Configures the "Paste Special" dialog.

Parent:

config

Children:

None.

Attributes:

prompt (true|false) "false"

If set to true, the "Paste special" dialog will be opened and present a selection of paste types whenever content is pasted from MS Word.

preselectedaction (PASTEWITHFILTER|PASTEPLAINTEXT|PASTE|PASTEWITHSTYLEDEFS) "PASTE"

This specifies which paste action will be selected by default in the "Paste special" dialog when it is opened.

pasteasimage (true|false) "false"

If set to true, content inserted from the clipboard can be inserted using an image data flavor, provided the application the content was copied from putsthis data flavor on the clipboard. Note that this will insert the data exactly as it is found on the clipboard, i.e. in some cases it is not recommended to insert structured content copied from e.g. Word as an image.

Example:

```
<pastespecial prompt="false" preselectedaction="PASTE">
```

preservestyle

Description:

Preserves the formatting of the current paragraph when it is split into a new paragraph by pressing enter. The new paragraph will have the same formatting as the original paragraph if this setting is enabled.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If set to true, the formatting of the paragraph at the caret location is preserved when it is split by pressing enter.

Example:

```
<preservestyle enabled="true" >
```

presetsymbols

Description:

Wrapper for the `symbol` element specifying custom symbols available in the "Insert Special Character" dialog.

Parent:

config

Children:

symbol

Attributes:

only (true|false) "false"

If `only` is set to true, only preset symbols specified by the integrator will be available in the "Insert Special Character" dialog.

Example:

```
<presetsymbols>
  <symbol entity="Α" />
  <symbol entity="Β" />
  <symbol entity="Γ" />
  <symbol entity="Δ" />
</presetsymbols>
```

symbol

Description:

Specifies a custom symbol available in the "Insert Special Character" dialog.

Parent:

config

Children:

symbol

Attributes:

entity CDATA #REQUIRED

Specifies a custom symbol available in the "Insert Special Character" dialog as entity. The symbol will then be displayed in the "Additional Symbols" tab of the "Insert Special Character" dialog.

Example:

```
<symbol entity="Δ" />
```

showall**Description:**

Determines whether the "Show all" mode is activated at the editor's start-up or not.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

Using this configuration setting overrides the value from the user preferences.

Example:

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

spellingcheckers**Description:**

Specifies the languages available in the spell checker. The spell checker languages are defined using the `spellingchecker` element. The element `addspellcheckwordurl` enables the user to add new words to a dictionary.

Parent:

config

Children:

spellingchecker, addspellcheckwordurl

Attributes:

autospellcheck (true|false) "false"

Determines whether the spell checker will check the document's spelling at run-time.

allowaddwords (true|false) "true"

Determines whether the user can add words to the user dictionary at run-time.

Example:

```
<spellingcheckers autospellcheck="true">
  <spellingchecker name="english"
    archive="lex/english.jar"
    default="true"
    version="100" />
  <addspellcheckwordurl url="http://myserver.com/adword.php" />
</spellingcheckers>
```

spellingchecker**Description:**

Specifies a language available in the spell checker.

Parent:

spellingcheckers

Children:

None.

Attributes:

name CDATA #REQUIRED

Determines the name of the language.

archive CDATA #REQUIRED

Specifies the location of the language JAR file.

properties CDATA #IMPLIED

Determines the location of the spell-checker properties file if specified. Per default, the properties file located in the spell-checker JAR file is used to set the spell-checker properties, but if this attribute set, the new properties file is used instead of the one in the JAR file for the specified language.

default (true|false) "false"

Determines whether the language is used as default language by the spell checker.

version CDATA #REQUIRED

The version is used when caching the spellchecker JAR on the client. The version may be updated in subsequent releases of edit-on Pro. If the version indicated here is newer than the version on client, the client will get the new version from the server and cache it locally on the client. If the version is the same as on the client, the version on the client will be loaded, thus reducing the loading time of edit-on Pro.

Example:

```
<spellingchecker name="english"
  archive="lex/english.jar"
  default="true"
  version="100" />
```

addspellcheckwordurl**Description:**

Specifies the URL of the server-side script that will receive the word data after clicking on the "Add word" button in the spell checker dialog. The request sending the word data will be a POST request. The word will be contained in a field called "WORD" while the language currently used will be stored in a field called "LANG". The server-side script should append the word to the corresponding dictionary file. The new word will be recognized the next time the spelling is checked. When this element is not set, clicking the "Add word" button will add the word to the user dictionary.

Parent:

spellingcheckers

Children:

None.

Attributes:

url

Specifies the URL of the server-side script that will receive the word data after selecting on the "Add word" button in the spell checker dialog.

Example:

```
<addspellcheckwordurl url="http://myserver.com/adword.php" />
```

sourceview**Description:**

Specifies whether the source view is available.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to `true`, the source view is available.

smartindent (true|false) "true"

Specifies whether smart indenting is enabled or not. Smart indenting enables the editor to export XHTML data in a clean, indented layout.

wordwrap (true|false) "false"

Determines whether word wrapping is active in source view.

readonly (true|false) "false"

Determines whether editing is possible in source view.

syntaxhighlighting (true|false) "false"

Specifies whether syntax highlighting is enabled in sourceview.

Note:

The wordwrap setting has no effect when syntax highlighting is active. The editor will always display a horizontal scrollbar instead.

fontname CDATA #IMPLIED

Specifies the font face used in source view.

Note:

This setting will only have an effect if the source view syntax highlighting is enabled.

fontsize CDATA #IMPLIED

Specifies the font size used in source view.

Note:

This setting will only have an effect if the source view syntax highlighting is enabled.

hidewysiwygtabwhendisabled (true|false) "false"

Specifies whether the WYSIWYG tab should be disabled when the Source View is disabled.

Example:

```
<sourceview enabled="true"
  smartindent="false"
  wordwrap="true"
  syntaxhighlighting="false" />
```

standalonemode**Description:**

Specifies whether the editor is displayed in frame window mode by default

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "false"

If `enabled` is set to true, the editor will be displayed in frame window mode by default.

Example:

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

tabpaneplacement**Description:**

Specifies whether the tab pane will be placed at the top or at the bottom of the editor.

Parent:

config

Children:

None.

Attributes:

isbottom (true|false) "false"

If `isbottom` is set to true, the tab pane will be placed at the bottom of the editor.

If `isbottom` is set to false, the tab pane will be placed at the top of the editor.

Example:

```
<tabpaneplacement isbottom="true" />
```

templateurl**Description:**

Specifies the location of the template loaded once the editor is initialized.

Parent:

config

Children:

None.

Attributes:

url CDATA #REQUIRED

Specifies the location of the template loaded once the editor is initialized. The location of the template may be relative to the editor's codebase.

Note:

The template will be loaded before any other content. It is also loaded before any of the events specified by the element `eopeventhandling` are fired. You can set a DocType in the template which will be used for any documents opened with the editor even if these documents specify other DocTypes, unless the `overridedoctype` attribute of the `documentvalidation` element is set to true. In this case, the editor will always use the DocType of the newly opened documents for validation.

Example:

```
<templateurl url="template.xhtml" />
```

text-antialiasing

Description:

Determines whether the text in the editor should be anti-aliased or not.

Parent:

config

Children:

None.

Attributes:

enabled (true|false|default) "default"

If set to true, text in the editor is anti-aliased. If "default" is used, the default alias setting of the system is used.

Example:

```
<text-antialiasing enabled="true" />
```

graphics-antialiasing

Description:

Determines whether images in the editor should be anti-aliased or not. If "default" is used, the default alias setting of the system is used.

Parent:

config

Children:

None.

Attributes:

enabled (true|false|default) "default"

If set to true, images in the editor are anti-aliased. If "default" is used, the default alias setting of the system is used.

Example:

```
<graphics-antialiasing enabled="true" />
```

thesaurus**Description:**

Specifies a dictionary for the thesaurus feature. Currently, only American English, British English and Canadian English are supported.

The thesaurus will only suggest words for a language if the corresponding dictionary is specified here.

Parent:

config

Children:

thesaurus

Attributes:

name CDATA #REQUIRED

Determines the name of the language.

archive CDATA #REQUIRED

Specifies the location of the language JAR file.

default (true|false) "false"

Determines whether the language is used as default language by the spell checker.

version CDATA #REQUIRED

The version is used when caching the spellchecker JAR on the client. The version may be updated in subsequent releases of edit-on Pro. If the version indicated here is newer than the version on client, the client will get the new version from the server and cache it locally on the client. If the version is the same as on the client, the version on the client will be loaded, thus reducing the loading time of edit-on Pro.

Example:

```
<thesauruses>
  <thesaurus name="LANGUAGE_AMERICAN" archive="lex/american-thesaurus.jar" version="100" default="t">
  <thesaurus name="LANGUAGE_BRITISH" archive="lex/british-thesaurus.jar" version="100" default="t">
  <thesaurus name="LANGUAGE_CANADIAN" archive="lex/canadian -thesaurus.jar" version="100" default="t">
</thesauruses>
```

thesauruses**Description:**

Specifies a list of dictionaries for the thesaurus feature. Currently, only American English and British English are supported.

The thesaurus will only suggest words for a language if the corresponding dictionary is specified here.

Parent:

config

Children:

thesaurus

Attributes:

None.

Example:

```
<thesauruses>
  <thesaurus name="LANGUAGE_AMERICAN" archive="lex/american-thesaurus.jar" version="100" default="t">
  <thesaurus name="LANGUAGE_BRITISH" archive="lex/british-thesaurus.jar" version="100" default="t">
  <thesaurus name="LANGUAGE_CANADIAN" archive="lex/canadian -thesaurus.jar" version="100" default="t">
</thesauruses>
```

uploadimagesmethod**Description:**

Wrapper for the elements `http` and `webdav` which specify the method that will be used to upload images to the server.

Parent:

`config`

Children:

`http` | `webdav`

Attributes:

None.

Example:

```
<uploadimagesmethod>
  <webdav username="root"
    userpassword="root"
    url="http://myserver.com/webdavfolder"/>
</uploadimagesmethod>
```

Important:

This setting is deprecated as of edit-on Pro version 4.2. Please use the setting `filemanagement` instead.

filemanagement**Description:**

Wrapper for the `file` element which is used to specify file repositories or upload handlers for the different file types that can be handled by edit-on Pro.

Parent:

`config`

Children:

`file`

Attributes:

None.

Example:

```
<filemanagement>
  <file type="image"
    baseurl="http://www.myserver.com/"
    autoupload="true"
    onuploadfinished="myEventHandler">
    <webdav url="http://www.myserver.com/"
      username="webdav"
      encryptedpassword="webdav="
      debug="true"
      putmethodexpectation="false"
      mimetype="image/gif" />
  </file>
</filemanagement>
```

file**Description:**

Determines how files of the specified type should be handled by edit-on Pro. If a WebDAV connection is specified, the files of the specified type can be inserted from or uploaded to the repository.

Parent:

file

Children:

http | webdav

Attributes:

type (image|object|document) #REQUIRED

Specifies the file type that will be managed. For details, please see the table "Overview of ACTIONS and their corresponding configuration" in the chapter [Uploading and Inserting Files Using WebDAV](#).

mimetype CDATA #IMPLIED

Allows to specify a list of mime types that will be used to determine which files will be displayed in the "Upload Images" and "Browse Image Repository" dialogs. E.g. mimetype="image/gif,image/jpeg" will restrict the type of images that will be displayed in these dialogs to images in JPEG and GIF format.

baseurl CDATA #IMPLIED

Specifies a base URL which will be used to convert absolute file paths to relative paths when uploading the files are uploaded to or inserted from the repository.

autoupload (true|false) false

If set to true, an upload dialog is opened for this file type when the `SAVE` action is triggered or the `postDocumentToServer` API function is called.

maxsize CDATA #IMPLIED

Specifies the maximal size of the files that can be uploaded (in bytes).

onuploadfinished CDATA #IMPLIED

Specifies a JavaScript event handler function that will be called after the upload is finished. The function will receive two arguments, `obj` and `status`. While `obj` contains a reference to the applet, `status` is a two-dimensional array containing information about the upload of each file. It will contain the following info for each file:

The old name of the file

The new name of the uploaded file

The new file path on the server

The status code returned by the server

The type of the upload that was performed (`image` , `object` or `document`)

Example:

```
<file type="image"
  baseurl="http://www.myserver.com/"
  autoupload="true"
  onuploadfinished="imageEventHandler">
```

http

Description:

Specifies the location the files of the specified type contained in the document will be posted to when the `corresponding` upload action is fired. Only files not located at the URL specified by the `base` element will be posted.

Parent:

`file`

Children:

None.

Attributes:

`url` CDATA #REQUIRED

Specifies the location the files contained in the document will be posted to when the corresponding action is fired.

`defaultdialog` (true|false) "true"

Specifies whether the "Image Upload" dialog should be displayed when uploading the specified file type. If set to false, all files of the specified type are uploaded when the file upload is started, and no dialog allowing the user to select individual files to upload is displayed.

Example:

```
<http url="http://myserver.com/imageupload.php" />
```

webdav**Description:**

Determines the location of the WebDAV folder that will serve as repository when uploading files corresponding to the file type specified by the parent `file` element.

Parent:

`file`

Children:

None.

Attributes:**Note:**

In case the value of the username attribute, the password attribute or both attributes are empty, a login dialog will be displayed. This dialog gives the user three attempts to authenticate to the WebDAV repository.

`username` CDATA #IMPLIED

Determines the name of the user to authenticate as when connecting to the WebDAV location specified by `url`.

`userpassword` CDATA #IMPLIED

Determines the password used to authenticate the user specified by `username` when connecting to the location specified by `url`.

Important:

The password for the WebDAV connection specified by the `userpassword` attribute may not contain whitespace. Please make sure it does not contain any whitespace or the connection to the WebDAV repository may fail if a password is required.

`encryptedpassword` CDATA #IMPLIED

Determines the password used to authenticate the user specified by `username` when connecting to the location specified by `url`. Unlike `userpassword`, this attribute will not take the password as is in plain text as argument, but rather the encrypted version of the password that was encrypted using the password encryption tool found in the `/tools` directory in the edit-on Pro installation package.

Important:

The password for the WebDAV connection specified by the `encryptedpassword` attribute may not contain whitespace. Please make sure it does not contain any whitespace or the connection to the WebDAV repository may fail if a password is required.

`url` CDATA #REQUIRED

Determines the location of the WebDAV folder images contained in the document will be uploaded to using the "Upload Images" dialog.

`debug` (true|false) "false"

If set to true all actions of the WebDAV connections are logged within the Java console.

`mimetype` CDATA #IMPLIED

If set, only files with the specified mime types will be displayed in the WebDAV dialogs.

`putmethodexpectation` (true|false) "true"

If set to false, the Expect: 100-continue handshake will not be used by the webdav client when sending a request to the server. Set this to false if your server does not fully support HTTP 1.1 or if you are receiving a server response with the HTTP error code 417 when using WebDAV.

`fullaccess` (true|false) "false"

If set to true, the user is allowed to create new directories, rename files and delete files in the WebDAV directory.

Note:

The purpose of the 100 (Continue) status (refer to section 10.1.1 of the RFC 2616 for more details) is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request headers) before the client sends the request body. In some cases, it might either be inappropriate or highly inefficient for the client to send the body if the server will reject the message without looking at the body.

Example:

```
<webdav username="webdavuser"
  userpassword="webdavuserpassword"
  url="http://myserver.com/webdavfolder/"
  debug="true"
  mimetype="image/gif,image/jpeg" />
```

uicolors**Description:**

Wrapper for the `uicolor` element, which allows to individually specify custom colors for each area of the editor's user interface.

Parent:

`config`

Children:

`uicolor`

Attributes:

None.

Example:

```
<uicolors>
  <uicolor name="windowfacecolor" rgb="#000099" />
  <uicolor name="tabpaneactivecolor" rgb="#eeeeee" />
  <uicolor name="windowhighlightcolor" rgb="#ffcc66" />
  <uicolor name="darkedgecolor" rgb="#ff0000" />
</uicolors>
```

uicolor**Description:**

Allows to specify custom colors for each area of the editor's user interface.

Parent:

`uicolors`

Children:

None.

Attributes:

`name` (windowfacecolor | tabpaneactivecolor | windowhighlightcolor | lightedgecolor | darkedgecolor | innertextcolor | disablediconcolor) #REQUIRED

Specifies the area of the user interface where the custom color should be used.

`rgb` CDATA #REQUIRED

Determines the hexadecimal RGB value of the custom color.

Example:

```
<uicolor name="windowfacecolor" rgb="#000099" />
```

userpreferences

Description:

Determines whether or not the saving of user preferences should be enabled or not. If user preferences are enabled, some dialogs will retain user-specified values such as custom colors, table size etc.

Parent:

config

Children:

None.

Attributes:

enabled (true|false) "true"

If set to true, user preferences will be stored and used by the editor.

Example:

```
<userpreferences enabled="false" />
```

3.2 JavaScript API

3.2.1 JavaScript Configuration API

Important:

The functions described in this chapter may only be used before initializing an instance of edit-on Pro as they are used to configure the applet. Calling the function `loadEditor` will start the initialization of the corresponding editor instance.

editOnPro

Syntax:

```
object editOnPro (
    string strWidth
    string strHeight
    string strName
    string strId
    string strObj
)
```

Description:

Constructor for the `editOnPro` object. Creates an editor object with the width `strWidth`, the height `strHeight`, the name `strName` and the id `strId`.

Important:

The value of `strObj` must always be the same as the name of the `editOnPro` object you are creating.

Example:

```
eop = new editOnPro(725, 350, "myEditor", "myId", "eop");
```

setCodeBase

Syntax:

```
void setCodeBase(  
    string strCodebase  
)
```

Description:

The URL location `strCodebase` is used as the editor's codebase.

Example:

```
eop.setCodeBase("http://myserver.com/EOP");
```

addArchive

Syntax:

```
void addArchive (  
    string strArchive  
    string strVersion  
)
```

Description:

Specifies a Java archive to be loaded in addition to the editor's archive. To load multiple additional archives, call this function for each archive. `strArchive` specifies the URL to the archive to be loaded. Use `strVersion` to specify a version number for the archive. If caching is enabled, this version number will be used when caching the archive.

Example:

```
eop.addArchive("myarchive.jar","1.0.8.0");
```

setArchive

Syntax:

```
void setArchive(  
    string strArchive  
)
```

Description:

Specifies the URL to the applet Java Archive.

Example:

```
eop.setArchive("edit-on-pro-4.jar");
```

setParam

Syntax:

```
void setParam(
    string strProperty
    string strValue
)
```

Description:

The parameter `strProperty` with the value `strValue` is added to the editor parameters.

Example:

```
eop.setParam("sourceview", "true");
```

setBodyOnly

Syntax:

```
void setBodyOnly(
    boolean bodyonly
)
```

Description:

Determines whether the "bodyonly" mode should be activated or not.

Example:

```
eop.disableCaching();
```

disableCaching

Syntax:

```
void disableCaching()
```

Description:

Disables the applet caching feature for this instance of the editor. Caching is enabled by default unless this method is called before initializing the editor.

Example:

```
eop.disableCaching();
```

setConfig

Syntax:

```
void setConfig(
    string sConfig
)
```

Description:

Specifies the editor's configuration as string.

Example:

```
strConf =
    ' <?xml version="1.0" encoding="UTF-8"?><config> ... </config> ';
eop.setConfig(strConf);
```

setConfigURL

Syntax:

```
void
    setConfigURL(
        string sURL
    )
```

Description:

Specifies the URL location of the editor's configuration file. May be relative to the applet's codebase.

Example:

```
eop.setConfigURL("config.xml");
```

setExportAsNumerical

Syntax:

```
void setExportAsNumerical(
    boolean bExportAsNum
)
```

Description:

Sets the value of the `exportasnumerical` property.

Example:

```
eop.setExportAsNumerical(true);
```

Note:

Characters will only be exported as entities if they are not supported by the encoding specified in the document. If characters are supported by the specified encoding, they will be exported as such.

setHeartBeatInterval

Syntax:

```
void setHeartBeatInterval(
    string strInterval
)
```

Description:

Specifies in which interval (in seconds) the editor will perform a HTTP request to the URL specified by `setHeartBeatInterval`. The default interval is 60 seconds.

Example:

```
eop.setHeartBeatInterval("30");
```

setHeartBeatURL

Syntax:

```
void setHeartBeatURL(
    string strURL
)
```

Description:

Specifies an URL the editor will perform a GET request to in the interval specified by `setHeartBeatInterval` or in an interval of 60 seconds if `setHeartBeatInterval` is not specified. This can e.g. be used to prevent a session from timing out if you are using sessions.

Example:

```
eop.setHeartBeatURL("http://yourserver.com/session.htm");
```

setHiddenLoading

Syntax:

```
void setHiddenLoading(
    boolean bState
    [boolean displayOnLoad]
    [int loadingIndicatorOffset]
    [int loadingIndicatorTimeout]
)
```

Description:

The argument `bState` specifies if the editor should be visible while it is loading. If set to `true`, an animated image will be displayed while the editor is loading.

`displayOnLoad` specifies if the editor should be displayed once it is fully loaded. If set to `false`, the editor will remain hidden until the method `showEditor` is called.

`loadingIndicatorOffset` specifies the time in milliseconds after which the progress indicator should be displayed from the moment on the editor has started loading.

`loadingIndicatorTimeout` specifies the time in milliseconds after which the progress indicator will not be displayed anymore if an error occurred when loading the editor.

Example:

```
eop.setHiddenLoading(true);
```

setLoadingIndicator

Syntax:

```
void setLoadingIndicator(
    string strIndicator
)
```

Description:

If hidden loading is enabled, this specifies the HTML code that is displayed while the editor is loading.

Example:

```
strLoad = "The editor is loading. Please wait.";
eop.setLoadingIndicator(strLoad);
```

setLicenseKey**Syntax:**

```
void setLicenseKey(
    string strLicenseKey
)
```

Description:

Sets the value of the `licensekey` property.

Example:

```
eop.setLicenseKey("licensekey.xml");
```

setLocaleCode**Syntax:**

```
void setLocaleCode(
    string strLocale
)
```

Description:

Sets the value of the `locale` property.

Example:

```
eop.setLocaleCode("en_US");
```

setLocaleURL**Syntax:**

```
void
    setLocaleURL(
        string strLocaleURL
    )
```

Description:

Sets the value of the `localeurl` property.

Example:

```
eop.setLocaleURL("locale_it_IT.xml");
```

setNewDocumentDialog**Syntax:**

```
void setNewDocumentDialog(
    boolean bNewDocDialog
)
```

Description:

Sets the value of the `newdocumentdialog` property.

Example:

```
eop.setNewDocumentDialog(false);
```

setStandAloneHeight

Syntax:

```
void setStandAloneHeight (  
    string sHeight  
)
```

Description:

Sets the default height of the editor in stand-alone mode in pixels.

Example:

```
eop.setStandAloneHeight ("500");
```

setStandAloneWidth

Syntax:

```
void setStandAloneWidth (  
    string sWidth  
)
```

Description:

Sets the default width of the editor in stand-alone mode in pixels.

Example:

```
eop.setStandAloneWidth ("500");
```

setStartUpScreenBackgroundColor

Syntax:

```
void setStartUpScreenBackgroundColor (  
    string strColor  
)
```

Description:

Specifies the background color of the applet during initialization.

Example:

```
eop.setStartUpScreenBackgroundColor ("#ffffff");
```

setStartUpScreenTextColor

Syntax:

```
void setStartUpScreenTextColor (  
    string strColor  
)
```

Description:

Specifies the color of the text displayed in the applet during initialization.

Example:

```
eop.setStartUpScreenTextColor ("#000000");
```

setUIConfig

Syntax:

```
void setUIConfig(
    string sUIConfig
)
```

Description:

Specifies the user interface configuration as string.

Example:

```
strUIConfig = ' <?xml version="1.0" encoding="UTF-8"?>
<uiconfig> <actions> ... </actions>
<menubar> ... </menubar> <toolbar> ...
</toolbar> <contextmenu> ... </contextmenu>
</uiconfig> ';
eop.setUIConfigURL("uiconfig.xml");
```

setUIConfigURL

Syntax:

```
void setUIConfigURL(
    string sURL
)
```

Description:

Specifies the URL location of the editor's user interface configuration file. May be relative to the applet's codebase.

Example:

```
eop.setUIConfigURL("uiconfig.xml");
```

setUnsupportedBrowserEvent

Syntax:

```
void setUnsupportedBrowserEvent(
    string sEvent
)
```

Description:

If this is specified, the function with the name `sEvent` will be called if the browser / OS combination that is used is not supported.

Example:

```
eop.setUnsupportedBrowserEvent("myFunc");
```

setIconRepositoryURL

Syntax:

```
void setIconRepositoryURL(
    string sIconRepositoryURL
)
```

Description:

Specifies a repository from which the editor loads the toolbar icons. If this repository is specified, the applet will first look for icons in the repository. If the icon can not be found in the repository, it is loaded from the edit-on Pro JAR file.

If a relative path is used for the repository, it is resolved relative to the editor's codebase.

Example:

```
eop.setIconRepositoryURL("myicons/");
```

setDefaultFont

Syntax:

```
void setDefaultFont(
    string sDefaultFont
)
```

Description:

Specifies the default font used to display the editor's content where no font is otherwise specified.

This font is only used to render the editor's content, and is not exported in any way.

Example:

```
eop.setDefaultFont("Arial");
```

setEventHandler

Syntax:

```
void setEventHandler(
    string sEvent, string sFunction
)
```

Description:

Calls the function sFunc every time the event sEvent is fired. For details about available events, please see the chapter [Event Handlers](#).

Example:

```
eop.setEventHandler("ONDATALOADED", "onDataLoaded");
```

loadEditor

Syntax:

```
void loadEditor()
```

Description:

Inserts and initializes the editor at the location in the document where this function is executed.

Example:

```
eop.loadEditor();
```

preloadApplet

Syntax:

```
void preloadApplet(
    string codebase
    boolean enableJREAutoDownload
    [string JREAutoDownloadURL]
    [string JREMinVersion]
)
```

Description:

This initializes the JVM and additionally preloads the applet JAR file into the JRE cache. This function should be used on a page before the editor is loaded in order to speed up subsequent start times of the editor.

Note:

The function `preloadApplet` can be used independently of the `editOnPro` class. It should be used on a page of your system which does not contain the editor itself, and before the editor is actually instantiated. It is not recommended to use this function on a page containing edit-on Pro.

Important

Using the preload mechanism may cause a warning dialog to be displayed if Java 1.7.0_11 or newer is used. From this Java version on, this dialog is displayed for all unsigned applets.

`codebase`

Specifies the location of the editor's codebase resp. of "preload.jar" which will be loaded to initialize the JVM.

`enableJREAutoDownload`

This will only have an effect in Internet Explorer. If set to true, the browser will try to install a JRE from the location specified by `JREAutoDownloadURL` and no Java was detected or the Java version is below the version specified by `JREMinVersion`.

`JREAutoDownloadURL` (optional)

This is only supported in Internet Explorer. If set and no JVM is installed, the browser will try to install a JVM from the specified location.

`JREMinVersion` (optional)

This is only supported in Internet Explorer. This parameter must be set if `JREAutoDownloadURL` was set. This specifies the minimum version of the JRE that should be installed.

Example:

```
preloadApplet("eopro",
    true,
    "http://java.sun.com/products/plugin/autodl/jinstall-1_4-windows-i586.cab",
    "1,4,2");
```

preloadJVM

Syntax:

```
void preloadJVM(
    string codebase
    boolean enableJREAutoDownload
    [string JREAutoDownloadURL]
    [string JREMinVersion]
)
```

Description:

This initializes the JVM. This function should be used on a page before the editor is loaded in order to initialize the Java VM. If the Java VM is already started before the editor is loaded, the startup time of the editor will seem faster to the editor, as the start of the Java VM itself can be quite time consuming on some systems.

Note:

The function `preloadJVM` can be used independently of the `editOnPro` class. It should be used on a page of your system which does not contain the editor itself, and before the editor is actually instantiated. Its only purpose is to ensure the JVM is already started when accessing a page with an instance of edit-on Pro. It is not recommended to use this function on a page containing edit-on Pro.

Important

Using the preload mechanism may cause a warning dialog to be displayed if Java 1.7.0_11 or newer is used. From this Java version on, this dialog is displayed for all unsigned applets.

`codebase`

Specifies the location of the editor's codebase resp. of "preload.jar" which will be loaded to initialize the JVM.

`enableJREAutoDownload`

This will only have an effect in Internet Explorer. If set to true, the browser will try to install a JRE from the location specified by `JREAutoDownloadURL` and no Java was detected or the Java version is below the version specified by `JREMinVersion`.

`JREAutoDownloadURL` (optional)

This is only supported in Internet Explorer. If set and no JVM is installed, the browser will try to install a JVM from the specified location.

`JREMinVersion` (optional)

This is only supported in Internet Explorer. This parameter must be set if `JREAutoDownloadURL` was set. This specifies the minimum version of the JRE that should be installed.

Example:

```
preloadJVM("eopro",
    true,
    "http://java.sun.com/products/plugin/autodl/jinstall-1_4-windows-i586.cab",
    "1,4,2");
```

3.2.2 JavaScript API / Java API

Important:

All functions (excepted `get` functions) described in this chapter will not be executed immediately as they are called. When they are called, they will first be queued and be executed in sequence when the method `pumpEvents()` is called.

acceptAll

Syntax:

```
void acceptAll()
```

Description:

Accepts all changes made in the document and ends comparison mode.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
eop.acceptAll();
```

addStyleSheet

Syntax:

```
void addStyleSheet(  
    string strStyles  
)
```

Description:

Loads styles from `strStyles` into the editor.

Example:

```
myStyles = "p { text-decoration:underline; }";  
eop.addStyleSheet(myStyles);
```

cleanStyleSheet

Syntax:

```
void cleanStyleSheet()
```

Description:

Erases any existing styles within the document. Styles available in the editor because of an external style sheet file will still be available.

Example:

```
eop.cleanStyleSheet();
```

clear

Syntax:

```
void clear()
```

Description:

Clears the editor's content.

Example:

```
eop.clear();
```

compareDocumentContents

Syntax:

```
void compareDocumentContents(
    string strDoc1,
    string strDoc2
)
```

Description:

Loads and compares two documents as string.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
strDoc1 = "<p>Some content</p>";
strDoc2 = "<p>Some more content</p>";
eop.compareDocumentContents(strDoc1, strDoc2);
```

compareDocumentContentToEditorContent

Syntax:

```
void compareDocumentContentToEditorContent(
    string strDoc
)
```

Description:

Compares the document specified by the string `strDoc` to the editor content.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
strDoc = "<p>Some content</p>";
eop.compareDocumentToEditorContent(strDoc1);
```

compareDocuments

Syntax:

```
void compareDocuments(
    string strURL1,
    string strURL2
)
```

Description:

Loads and compares two documents from an URI.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
strURL1 = "http://www.yourserver.com/doc1.htm";
strURL2 = "http://www.yourserver.com/doc2.htm";
eop.compareDocuments(strURL1, strURL2);
```

compareDocumentToEditorContent**Syntax:**

```
void compareDocumentToEditorContent(
    string strURL
)
```

Description:

Compares the document from the specified URL `strURL` to the editor content.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
strURL = "http://www.myserver.com/doc.htm";
eop.compareDocumentToEditorContent(strDoc1);
```

decode**Syntax:**

```
string decode(
    string strEncodedString
)
```

Description:

Decodes all HTML entities contained in the specified string to character data and returns the decoded string.

Example:

```
strEncodedString = "&";
strDecodedString = eop.decode(strEncodedString); //This will return "&"
```

encode**Syntax:**

```
string encode(
    string strDecodedString
)
```

Description:

Encodes the specified string according to the encoding specified in the applet by the `encoding` property. If no encoding is specified, the string will be HTML encoded.

Example:

```
strDecodedString = "&";
strEncodedString = eop.encode(strDecodedString); //This will return "&"
```

enforceReadOnlyAPI

Syntax:

```
void enforceReadOnlyAPI(
    boolean bEnforce
)
```

Description:

Toggles whether or not the JavaScript API methods can modify the content of read-only custom elements (i.e. any custom element where any of the custom CSS properties "ro-readonlycontent", "ro-readonlyelement" or "ro-formattingonly" are set to true).

If set to "false", it is possible to use JavaScript API methods to modify the content of read-only areas.

Setting `bEnforce` to true prevents read-only areas from being modified by JavaScript API methods (such as `setHTMLData`).

Example:

```
eop.enforceReadOnlyAPI(false);
eop.pumpEvents();
```

getActiveSpellChecker

Syntax:

```
string getActiveSpellChecker()
```

Description:

Returns the `localeid` of the currently active spell-checker language. The `localeid` is used in the XML locale file of the editor to map the strings used in dialogs and the `localeid` attributes of the elements defined in the user interface configuration file to the localized strings of the corresponding language, e.g. "LANGUAGE_AMERICAN" corresponds to "American English".

Example:

```
strLang = eop.getActiveSpellChecker();
```

getBase

Syntax:

```
string getBase()
```

Description:

Returns the value of the `base` property.

Example:

```
base = eop.getBase();
```

getBlockElementAttributes

Syntax:

```
Array getBlockElementAttributes()
```

Description:

Returns an associative array containing the attributes of the current block level element (e.g. "<p>" or "<div>") and their values. The attribute names of the element form the keys of the array returned by the function, while the attribute values are its values.

Example:

```
arr = eop.getBlockElementAttributes();
msg = "";
for (var prop in arr) {
    msg += prop+'="'+arr[prop]+'"\n';
}
if(msg.length<1) msg = "No attributes";
alert(msg);
```

getCharNoSpacesStatistic

Syntax:

```
string getCharNoSpacesStatistic()
```

Description:

Retrieves the number of characters contained in the document. White spaces are ignored.

Example:

```
noSpacesStat = eop.getCharNoSpacesStatistic();
```

getBookmarks

Syntax:

```
Array getBookmarks(boolean bSorted)
```

Description:

Returns an array containing the names of all bookmarks in the document. The boolean parameter `bSorted` indicates if this array should be sorted according to the position where the bookmark occurs in the document.

Example: Displaying all bookmarks in the document.

```
function getBookmarks() {
    arr = eop.getBookmarks(true);
    msg = "";
    for (var prop in arr) {
        msg += prop+'="'+arr[prop]+'"\n';
    }
    if(msg.length<1) msg = "No bookmarks";
    alert(msg);
}
```

getCharWithSpacesStatistic

Syntax:

```
string getCharWithSpacesStatistic()
```

Description:

Retrieves the number of characters contained in the document including white spaces.

Example:

```
charStat = eop.getCharWithSpacesStatistic();
```

getCurrentElement

Syntax:

```
string getCurrentElement()
```

Description:

Retrieves the element at the current cursor position, including the textual content as well as all children elements.

Example:

```
curElement = eop.getCurrentElement();
```

getCurrentElementContent

Syntax:

```
string getCurrentElementContent()
```

Description:

Retrieves the content of the element at the cursor position, including the textual content as well as all child elements.

Example:

```
curElementContent = eop.getCurrentElementContent();
```

getCurrentTableColumnCellAttributes

Syntax:

```
Array getCurrentTableColumnCellAttributes()
```

Description:

Returns the attributes of all table cells in the current table column. The array that is returned is a multi-dimensional array consisting of an array of all table cells of the current column and an array of all attributes for each cell.

Example:

```
function alertTableColumnCellAttributes() {
    arr = eop.getCurrentTableColumnCellAttributes();
    str = "";
    for (key in arr) {
        str += "Cell: " + key + " Attributes: {";
        for (innerKey in arr[key]) {
            str += "\n\t" + innerKey + ": " + arr[key][innerKey];
        }
        str += "\n}\n";
    }
    alert(str);
}
```

getCurrentTableRowCellAttributes

Syntax:

```
Array getCurrentTableRowCellAttributes()
```

Description:

Returns the attributes of all table cells in the current table row. The array that is returned is a multi-dimensional array consisting of an array of all table cells of the current row and an array of all attributes for each cell.

Example:

```
function alertTableRowCellAttributes() {
    arr = eop.getCurrentTableRowCellAttributes();
    str = "";
    for (key in arr) {
        str += "Cell: " + key + " Attributes: {";
        for (innerKey in arr[key]) {
            str += "\n\t" + innerKey + ": " + arr[key][innerKey];
        }
        str += "\n}\n";
    }
    alert(str);
}
```

getCustomField

Syntax:

```
string getCustomField()
```

Description:

Retrieves the value of the `customfield` property.

Example:

```
customField = eop.getCustomField();
```

getDTDBase

Syntax:

```
string getDTDBase()
```

Description:

Retrieves the base URL that is used to resolve relative DTDs if it is set.

Example:

```
strBase = eop.getDTDBase();
```

getElementAttributes

Syntax:

```
Array getElementAttributes()
```

Description:

Returns an associative array containing the attributes of the current element and their values. The attribute names of the element form the keys of the array returned by the function, while the attribute values are its values.

Example:

```
arr = eop.getElementAttributes();
msg = "";
for (var prop in arr) {
    msg += prop + '=' + arr[prop] + '\n';
}
if(msg.length<1) msg = "No attributes";
alert(msg);
```

getEncoding

Syntax:

```
string getEncoding()
```

Description:

Retrieves the value of the `encoding` property.

Example:

```
curEncoding = eop.getEncoding();
```

getExportAsNumerical

Syntax:

```
boolean getExportAsNumerical()
```

Description:

Retrieves the value of the `exportasnumerical` property.

Example:

```
bExportAsNumerical = eop.getExportAsNumerical();
```

getGetErrorTarget

Syntax:

```
string getGetErrorTarget()
```

Description:

Retrieves the value of the `geterrortarget` property.

Example:

```
curGetErrorTarget = eop.getGetErrorTarget();
```

getGetErrorURL

Syntax:

```
string getGetErrorURL()
```

Description:

Retrieves the value of the `geterrorurl` property.

Example:

```
curGetErrorURL = eop.getGetErrorURL();
```

getGetHTMLDataURL

Syntax:

```
string getGetHTMLDataURL()
```

Description:

Retrieves the value of the `gethtmldataurl` property.

Example:

```
curGetHTMLDataURL = eop.getGetHTMLDataURL();
```

getGetOKTarget

Syntax:

```
string getGetOKTarget()
```

Description:

Retrieves the value of the `getoktarget` property.

Example:

```
curGetOKTarget = eop.getGetOKTarget();
```

getGetOKURL

Syntax:

```
string getGetOKURL()
```

Description:

Retrieves the value of the `getokurl` property.

Example:

```
curGetOKURL = eop.getGetOKURL();
```

getHeadElementContent

Syntax:

```
string getHeadElementContent()
```

Description:

Retrieves the content of the document's `<head>...</head>` element. This has no effect if the `bodyonly` property is set to "true".

Example:

```
headElementContent = eop.getHeadElementContent();
```

getHTMLData

Syntax:

```
string getHTMLData()
```

Description:table_numcols

Retrieves all content from the editor.

Example:

```
strContent = eop.getHTMLData();
```

getImageBase

Syntax:

```
string getImageBase()
```

Description:

Returns the value of the `imagebase` property.

Example:

```
imgBase = eop.getImageBase();
```

Important:

This method is deprecated as of edit-on Pro version 4.2. Please use `automaticallygetBase` instead.

getImagesStatistic

Syntax:

```
string getImagesStatistic()
```

Description:

Retrieves the number of images contained in the document.

Example:

```
noImages = eop.getImagesStatistic();
```

getLinesStatistic

Syntax:

```
string getLinesStatistic()
```

Description:

Retrieves the number of lines contained in the document.

Example:

```
noLines = eop.getLinesStatistic();
```

getLocaleCode

Syntax:

```
string getLocaleCode()
```

Description:

Retrieves the value of the `locale` property.

Example:

```
curLocale = eop.getLocaleCode();
```

getLocaleFromBrowser

Syntax:

```
string getLocaleFromBrowser()
```

Description:

Returns the locale of the browser as string which can be used to set the editor's locale via `setLocaleCode` property. This will set the editor's locale to the browser's locale.

Example:

```
eop.setLocaleCode(eop.getLocaleCode());
```

getLocaleURL**Syntax:**

```
string getLocaleURL()
```

Description:

Retrieves the value of the `localeurl` property.

Example:

```
curLocaleURL = eop.getLocaleURL();
```

getNewDocumentDialog**Syntax:**

```
boolean getNewDocumentDialog()
```

Description:

Retrieves the value of the `newdocumentdialog` property.

Example:

```
bDocDialog = eop.getNewDocumentDialog();
```

getParagraphsStatistic**Syntax:**

```
string getParagraphsStatistic()
```

Description:

Retrieves the number of paragraphs contained in the document.

Example:

```
noParas = eop.getParagraphsStatistic();
```

getParentElementByName**Syntax:**

```
string getParentElementByName(  
    string strParent  
)
```

Description:

Starting from the caret position, searches for a parent element called `strParent` and retrieves the element, including the textual content as well as all children elements.

Example:

```
parentElement = eop.getParentElementByName("p");
```

getParentElementByNameContent

Syntax:

```
string getParentElementByNameContent (
    string strParent
)
```

Description:

Starting from the caret position, searches for a parent element called `strParent` and retrieves the textual content as well as all children elements of the element.

Example:

```
parentElementContent = eop.getParentElementByNameContent ("p");
```

getParentElementByNameAttributes

Syntax:

```
string getParentElementByNameAttributes (
    string strParent
)
```

Description:

Starting from the caret position, searches for a parent element called `strParent` and retrieves all attributes of the element as an array.

Example:

```
attributes = eop.getParentElementByNameAttributes ("p");
```

getPlainText

Syntax:

```
string getPlainText()
```

Description:

Retrieves the content from the editor as plain text without any tags. Line breaks ("`<br/>`") and paragraphs ("`<p></p>`") will be transformed to CRLF while all other tags are removed.

Example:

```
strPlainText = eop.getPlainText();
```

getPostErrorMessage

Syntax:

```
string getPostErrorMessage()
```

Description:

If an error occurred when using the direct HTTP POST API to post the editor's content, this method returns `null` or the error returned by the server-side script that was the target of the POST operation.

See [Direct HTTP Connection API](#) for more details.

Example:

```
strError = eop.getPostErrorMessage();
```

getReadOnly

Syntax:

```
boolean getReadOnly()
```

Description:30

Returns whether the editor is currently in read-only state. Returns `true` if the editor currently is in read-only state and can not be edited.

Example:

```
bState = eop.getReadOnly();
```

getSelectedHTMLData

Syntax:

```
string getSelectedHTMLData()
```

Description:

Retrieves the selected HTML data from the editor.

Example:

```
sel = eop.getSelectedHTMLData();
```

getSetErrorTarget

Syntax:

```
string getSetErrorTarget()
```

Description:

Retrieves the value of the `seterrortarget` property.

Example:automatically

```
setErrTarget = eop.getSetErrorTarget();
```

getSetErrorURL

Syntax:

```
string getSetErrorURL()
```

Description:

Retrieves the value of the `seterrorurl` property.

Example:

```
curSetErrorURL = eop.getSetErrorURL();
```

getSetOKTarget

Syntax:

```
string getSetOKTarget()
```

Description:

Retrieves the value of the `setoktarget` property.

Example:

```
curSetOKTarget = eop.getSetOKTarget();
```

getSetOKURL

Syntax:

```
string getSetOKURL()
```

Description:

Retrieves the value of the `setokurl` property.

Example:

```
curSetOKURL = eop.getSetOKURL();
```

getSourceView

Syntax:

```
boolean getSourceView()
```

Description:

Retrieves the value of the `sourceview` property.

Example:

```
bSourceView = eop.getSourceView();
```

getStandAloneMode

Syntax:

```
boolean getStandAloneMode()
```

Description:

Returns the value of the `standalonemode` property.

Example:

```
bStMode = eop.getStandAloneMode();
```

getStatistics

Syntax:

```
Array getWordsStatistic()
```

Description:

Returns an associative array containing the number of images, lines, paragraphs, characters without spaces, characters with spaces and words contained in the document. The following keys are available in the associative array returned by `getWordStatistics`: `images`, `lines`, `paragraphs`, `charNoSpaces`, `charWithSpaces`, `words`.

Example:

```
arr = eop.getStatistics();
msg = "";
for (var prop in arr) {
    msg += prop+": "+arr[prop]+"\\n";
}
alert(msg);
```

getWebDAVCookieString

Syntax:

```
Array getWebDAVCookieString()
```

Description:

Retrieves an associative array containing available Webdav cookies in case the user has previously logged into the Webdav repository using one of edit-on Pro's Webdav dialogs.

Example:

```
cookie = eop.getWebDAVCookieString();
```

getWordsStatistic

Syntax:

```
string getWordsStatistic()
```

Description:

Retrieves the number of words contained in the document.

Example:

```
noWords = eop.getWordsStatistic();
```

hasContentChanged

Syntax:

```
boolean hasContentChanged()
```

Description:

Retrieves the state of the document in the editor. Returns true if the document if the document was modified and returns false if it was not modified. The state of `hasContentChanged()` is reset after calling `getHTMLData()`.

Example:

```
bChanged = eop.hasContentChanged();
```

jumpToBookmark

Syntax:

```
void jumpToBookmark(
    string sName
    int mode
)
```

Description:

Jumps to a specific bookmark in the document.

The parameter `sName` is the name of the bookmark the caret should jump to.

The parameter `mode` is the type of jump to perform. There are three types of jump: you can select the bookmark, place the caret before the bookmark, or place the caret after the bookmark.

The constants available for the `mode` parameter are (assuming your edit-on Pro JavaScript object is called "eop"):

`eop.JUMPTOBOOKMARK_SELECT`: jumps to the bookmark and selects it.

`eop.JUMPTOBOOKMARK_BEFORE`: places the caret before the bookmark.

`eop.JUMPTOBOOKMARK_AFTER`: places the caret after the bookmark.

Example:

```
eop.jumpToBookmark("daffodil", eop.JUMPTOBOOKMARK_SELECT);
eop.pumpEvents();
```

insertCustomTag

Syntax:

```
void insertCustomTag(
    string strTagName
    boolean bEmpty
    string strAttributes
)
```

Description:

If `bEmpty` is set to true, the empty element `strTagName` is inserted at the cursor position.

If `bEmpty` is set to false, the element `strTagName` is inserted around the current selection. The element will have the attributes specified by `strAttributes`.

Example:

```
eop.insertCustomTag("mycustomtag", false, "myattrib='somevalue'");
```

insertImage

Syntax:

```
void insertImage(
    string strSrc
    string strAlt
    string strWidth
    string strHeight
    string strBorder
)
```

Description:

Inserts an image (GIF, JPG or PNG format) from the URL `strSrc` at the current cursor location.

`strWidth`, `strHeight` and `strBorder` specify the image's width, height and border width resp. `strAlt` specifies the image's ALT text.

Example:

```
strSrc = "http://www.google.com/images/logo.gif";
strWidth = "200";
strHeight = "100";
strBorder = "1";
strAlt = "some text describing the image...";
eop.insertImage(strSrc, strAlt, strWidth, strHeight, strBorder);
```

insertHTMLData

Syntax:

```
void insertHTMLData(
    string strContents
)
```

Description:

Inserts `strContents` at the cursor position.

Example:

```
strContents = "<p>Paragraph with
<strong>strong</strong> text</p>";
eop.insertHTMLData(strContents);
```

insertHTMLDataFromURL

Syntax:

```
void insertHTMLDataFromURL(
    string strURL
)
```

Description:

Inserts content from `strURL` at the cursor position.

Example:

```
eop.insertHTMLDataFromURL("http://www.w3.org");
```

invokeAction

Syntax:

```
void invokeAction(
    string actionID,
    [string selectedItem],
    [string eventHandler]
)
```

Description:

This emulates the clicking of the action `actionID` in the applet's menu bar, tool bar or context menu. If the action that should be fired corresponds to a drop-down box, the optional parameter `selectedItem` is used to select a specific item from the drop-down box. For a description of all available actions see the chapter [Actions](#).

The parameter `eventHandler` specifies an event handler function which is called once the action specified by the first parameter `actionID` has finished executing. If no event handler should be called, pass `null` as argument to this method.

Example:

```
eop.invokeAction("BOLD");
eop.invokeAction("BOLD", null, "myHandler");
eop.invokeAction("ADVANCEDSTYLESHEET", "H1");
eop.invokeAction("ADVANCEDSTYLESHEET", "H1", "myHandler");
```

isActionEnabled

Syntax:

```
boolean isActionEnabled()
```

Description:

Returns whether an edit-on Pro user interface action such as "BOLD", "ITALIC", ect. is enabled or not. If the action is enabled, it can be used from within the user interface. If it is disabled, every control using the action will be disabled and greyed out.

Example:

```
bIsActive = eop.isActionEnabled();
```

isVersioTrackEnabled

Syntax:

```
boolean isVersioTrackEnabled()
```

Description:

Returns whether or not VersioTrack is enabled in this instance of edit-on Pro.

Example:

```
bIsEnabled eop.isVersioTrackEnabled();
```

postDocumentToServer

Syntax:

```
void postDocumentToServer()
```

Description:

Performs a POST request to the URL specified by the `gethtmldataurl` property. The editor's content is stored in a field called `HTMLDATA`.

Example:

```
eop.postDocumentToServer();
```

rejectAll

Syntax:

```
void rejectAll()
```

Description:

Rejects all changes made in the document and ends comparison mode.

This only has an effect if VersioTrack is enabled in your license.

Example:

```
eop.rejectAll();
```

removeCurrentTableColumnCellAttributes

Syntax:

```
void removeCurrentTableColumnCellAttributes(
    Array attributes
)
```

Description:

Removes all attributes of all cells of the current table column if the attribute name corresponds to an attribute name given in the array.

Example:

```
function removeTableColumnCellAttributes() {
    @arr = new Array("height","width");
    @op.removeCurrentTableColumnCellAttributes(arr);
    @op.pumpEvents();
}
```

removeCurrentTableRowCellAttributes**Syntax:**

```
void removeCurrentTableRowCellAttributes (
    Array attributes
)
```

Description:

Removes all attributes of all cells of the current table row if the attribute name corresponds to an attribute name given in the array.

Example:

```
function removeTableRowCellAttributes() {
    arr = new Array("height", "width");
    eop.removeCurrentTableRowCellAttributes(arr);
    eop.pumpEvents();
}
```

removeParentElementByNameAttributes**Syntax:**

```
void removeParentElementByNameAttributes (
    String strParent
    Array arrAttribs
)
```

Description:

Searches for a parent element specified by `strParent` from the caret position. If a parent with this name is found, removes the attributes specified in `arrAttribs` from the element.

Example:

```
arr = new Array();
arr[0] = "style";
arr[1] = "id";

eop.removeParentElementByNameAttributes("p", arr);
eop.pumpEvents();
```

setActionEnabled**Syntax:**

```
void setActionEnabled(
    string strAction
    boolean bState
)
```

Description:

Sets the state of an edit-on Pro user interface action such as "BOLD", "ITALIC", ect. If set to true, the action is enabled and can be used from within the user interface. If it is set to false, the action is disabled. If it is integrated in the user interface, every control using the action will be disabled and greyed out.

Example:

```
eop.setActionEnabled("BOLD", false);
```

setActiveSpellChecker

Syntax:

```
string setActiveSpellChecker(string strLang)
```

Description:

Sets the `localeid` of the currently active spell-checker language. The `localeid` is used in the XML locale file of the editor to map the strings used in dialogs and the `localeid` attributes of the elements defined in the user interface configuration file to the localized strings of the corresponding language, e.g. "LANGUAGE_AMERICAN" corresponds to "American English".

The following spell-checker language localeids are available per default in the editor:

- American English: LANGUAGE_AMERICAN
- British English: LANGUAGE_BRITISH
- Canadian English: LANGUAGE_CANADIAN
- German: LANGUAGE_GERMAN
- French: LANGUAGE_FRENCH
- Spanish: LANGUAGE_SPANISH
- Italian: LANGUAGE_ITALIAN
- Dutch: LANGUAGE_DUTCH
- Swedish: LANGUAGE_SWEDISH
- Finish: LANGUAGE_FINISH
- Norwegian: LANGUAGE_NORWEGIAN
- Danish: LANGUAGE_DANISH
- Brazilian Portuguese: LANGUAGE_BRAZILIAN_PORTUGUESE
- Portuguese: LANGUAGE_PORTUGUESE
- American Medical: LANGUAGE_AMERICAN_MEDICAL
- British Medical: LANGUAGE_BRITISH_MEDICAL
- American Legal: LANGUAGE_AMERICAN_LEGAL
- British Legal: LANGUAGE_BRITISH_LEGAL

Example:

```
eop.setActiveSpellChecker("LANGUAGE_AMERICAN");
```

setBase

Syntax:

```
void setBase(
    string strBase
)
```

Description:

Sets the value of the `base` property.

Example:

```
eop.setBase("http://www.w3.org");
```

setBlockElementAttributes

Syntax:

```
void setBlockElementAttributes(
    Array arrAttribs
)
```

Description:

Sets the attributes and attribute values of the current block level element (e.g. "<p>" or "<div>"). Use an associative array containing key-value pairs of the attributes and their values to set the attributes.

Example:

```
arr = new Array();
arr["style"] = "color: red;";
arr["align"] = "left";
eop.setBlockElementAttributes(arr);
```

setCaretAtDocumentStart

Syntax:

```
void setCaretAtDocumentStart()
```

Description:

Sets the caret at the start of the document inside the editor.

Example:

```
eop.setCaretAtDocumentStart();
```

setCaretAfterNextBookmark

Syntax:

```
void setCaretAfterNextBookmark()
```

Description:

Sets the caret at the end of the next bookmark found in the document starting from the caret position.

Example:

```
eop.setCaretAfterNextBookmark();
```

setCaretAfterPreviousBookmark

Syntax:

```
void setCaretAfterPreviousBookmark()
```

Description:

Sets the caret at the end of the previous bookmark found in the document starting from the caret position.

Example:

```
eop.setCaretAfterPreviousBookmark();
```

setCurrentElementContent**Syntax:PlainText**

```
void setCurrentElementContent(
    string strContent
)
```

Description:

Sets the content of the tag at the current cursor position.

Example:

```
eop.setCurrentElementContent("the new content");
```

setCurrentTableColumnCellAttributes**Syntax:**

```
void setCurrentTableColumnCellAttributes(
    Array attributes
)
```

Description:

Sets the attributes of all cells of the current table column. If an attribute is already present in a table cell of the current column, its value is overwritten with the new value.

Example:

```
function setTableColumnCellAttributes() {
    arr = new Array("height");
    arr["height"] = 10;
    @op.setCurrentTableColumnCellAttributes(arr);
    @op.pumpEvents();
}
```

setCurrentTableRowCellAttributes**Syntax:**

```
void setCurrentTableRowCellAttributes(
    Array attributes
)
```

Description:

Sets the attributes of all cells of the current table row. If an attribute is already present in a table cell of the current row, its value is overwritten with the new value.

Example:

```
function setTableRowCellAttributes() {
    arr = new Array("height");
    arr["height"] = 10;
    @op.setCurrentTableRowCellAttributes(arr);
    @op.pumpEvents();
}
```

setCustomField

Syntax:

```
void setCustomField(  
    string strFieldValue  
)
```

Description:

Sets the value of the `customfield` property.

Example:

```
eop.setCustomField("myvalue");
```

setDTDBase

Syntax:

```
void setDTDBase(  
    string strBase  
)
```

Description:

Specifies the base URL which is used by the editor to resolve the DTDs with a relative path.

Example:

```
eop.setDTDBase("http://www.myserver.com/dtds/");
```

setEncoding

Syntax:

```
void setEncoding(  
    string strEncoding  
)
```

Description:

Sets the encoding used by the editor.

Example:

```
eop.setEncoding("UTF8");
```

setErrorTarget

Syntax:

```
void setErrorTarget(  
    string strGetErrTarget  
)
```

Description:

Sets the value of the `geterrortarget` property.

Example:

```
eop.setErrorTarget("_new");
```

setErrorURL**Syntax:**

```
void setErrorURL(
    string strGetErrURL
)
```

Description:

Sets the value of the `getErrorurl` property.

Example:

```
eop.setErrorURL("_new");
```

setGetHTMLDataURL**Syntax:**

```
void setGetHTMLDataURL(string strGetHTMLDataURL)
```

Description:

Sets the value of the `gethtmldataurl` property.

Example:

```
eop.setGetHTMLDataURL("http://www.realobjects.com");
```

setHeadElementContent**Syntax:**

```
void setHeadElementContent(string strContent)
```

Description:

Sets the document's `<head>...</head>` element content.

Example:

```
headElmContent =
    "<meta http-equiv='Content-type' content='text/html' />";
eop.setHeadElementContent(headElmContent);
```

setGetOKTarget**Syntax:**

```
void setGetOKTarget(
    string strGetOKTarget
)
```

Description:

Sets the value of the `getoktarget` property.

Example:

```
eop.setGetOKTarget("_blank");
```

setGetOKURL**Syntax:**

```
void setGetOKURL(
    string strGetOKURL
)
```

Description:

Sets the value of the `getokurl` property.

Example:

```
eop.setGetOKURL("http://myserver.com/ok.htm");
```

setHelpURL**Syntax:**

```
void setHelpURL(
    string sURL
)
```

Description:

Specifies the URL that will opened if the `HELP` action is called. If this method is not used, the online help corresponding to the `localeCode` specified in the editor is opened.

Example:

```
setHelpURL("http://www.yourserver.com/userguide.html");
```

setHTMLData**Syntax:**

```
void setHTMLData(
    string strContent
)
```

Description:

Loads `strContent` into the editor.

Example:

```
strContent =
"<p>Paragraph with <strong>strong</strong> text</p>";
eop.setHTMLData(strContent);
```

setHTMLDataFromURL**Syntax:**

```
void setHTMLDataFromURL(
    string strURL
)
```

Description:

Loads a HTML page from `strURL` into the editor.

Example:

```
eop.setHTMLDataFromURL("http://www.w3.org");
```

setElementAttributes**Syntax:**

```
void setElementAttributes(
    Object attribs
)
```

Description:

Sets the attributes and attribute values of the current element. Use an object containing key-value pairs of the attributes and their values to set the attributes.

Example:

```
obj = new Object();
obj["width"] = "100";
obj["height"] = "200";
obj["src"] = "http://myserver/image.gif";
eop.setElementAttributes(obj);
```

setIgnoreFilter**Syntax:**

```
void setIgnoreFilter(
    string strFilter
)
```

Description:**Example:**

```
eop.setIgnoreFilter(filter);
```

setImageBase**Syntax:**

```
void setImageBase(
    string strBase
)
```

Description:

Specifies a filter for elements which will be ignored in comparison mode. For details, see [Ignoring Changes](#)

Example:

```
eop.setImageBase("http://www.w3.org");
```

Important:

This method is deprecated as of edit-on Pro version 4.2. Please use `setBase` instead.

setLookAndFeel

Syntax:

```
void setLookAndFeel(
    string strLookAndFeel
)
```

Description:

Sets the look and feel used by the editor. The look and feels supported by default by the editor are:

Motif Look and Feel:

```
com.sun.java.swing.plaf.motif.MotifLookAndFeel
```

Windows Look and Feel:

```
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```

Mac Aqua Look and Feel:

```
apple.laf.AquaLookAndFeel
```

Metal Look and Feel:

```
javax.swing.plaf.metal.MetalLookAndFeel
```

Example:

```
strPlaf = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
eop.setLookAndFeel(strPlaf);
```

setNextBookmarkWrapAroundEnabled

Syntax:

```
void setNextBookmarkWrapAroundEnabled(
    boolean bEnabled
)
```

Description:

Enables wrap around when searching for the next bookmark in a document using the

`setCaretAfterNextBookmark` API method or the `GOTONEXTBOOKMARK` action.

Example:

```
eop.setNextBookmarkWrapAroundEnabled();
```

setParentElementByNameAttributes

Syntax:

```
void setParentElementByNameAttributes(
    String strParent
    Object attribs
)
```

Description:

Searches for a parent element specified by `strParent` from the caret position. If a parent with this name is found, sets the attributes and attribute values of the element. Use an object containing key-value pairs of the attributes and their values to set the attributes.

Example:

```
obj = new Object();
obj["style"] = "color: red;";
obj["lang"] = "en";
eop.setParentElementByNameAttributes("p", obj);
```

setParentElementByNameContent**Syntax:**

```
void setParentElementByNameContent (
    String strParent
    String strContent
)
```

Description:

Searches for a parent element specified by `strParent` from the caret position. If a parent with this name is found, the content of the element is replaced with the content specified by `strContent`.

Example:

```
eop.setParentElementByNameContent("p", "<span>new content</span>");
```

setPreviousBookmarkWrapAroundEnabled**Syntax:**

```
void setPreviousBookmarkWrapAroundEnabled(
    boolean bEnabled
)
```

Description:

Enables wrap around when searching for the previous bookmark in a document using the `setCaretAfterPreviousBookmark` API method or the `GOTOPREVIOUSBOOKMARK` action.

Example:

```
eop.setPreviousBookmarkWrapAroundEnabled();
```

setReadOnly**Syntax:**

```
void setReadOnly(
    boolean bState
)
```

Description:

Toggles editing in the editor. If `setReadOnly` is set to true, no editing actions can be performed anymore in the editor.

Note:

The JavaScript API functions will still function normaly when editing is disabled.

Example:

```
eop.setReadOnly(true);
```

setSetErrorTarget

Syntax:

```
void setSetErrorTarget(  
    string strSetErrorTarget  
)
```

Description:

Sets the value of the `seterrortarget` property.

Example:

```
eop.setSetErrorTarget("_blank");
```

setSetErrorURL

Syntax:

```
void setSetErrorURL(  
    string strSetErrorURL  
)
```

Description:

Sets the value of the `seterrorurl` property.

Example:

```
eop.setSetErrorURL("http://myserver.com/error.htm");
```

setSetOKTarget

Syntax:

```
void setSetOKTarget(  
    string strSetOKTarget  
)
```

Description:

Sets the value of the `setoktarget` property.

Example:

```
eop.setSetOKTarget("_new");
```

setSetOKURL

Syntax:

```
void setSetOKURL(  
    string strSetOKURL  
)
```

Description:

Sets the value of the `setokurl` property.

Example:

```
eop.setSetOKURL("http://myserver.com/ok.htm");
```

setSourceView

Syntax:

```
void setSourceView(  
    boolean bSourceView  
)
```

Description:

If set to `false`, the "Source View" tab of the editor is hidden. If set to `true`, the "Source View" tab is displayed again.

Example:

```
eop.setSourceView(false);
```

setStandAloneMode

Syntax:

```
void setStandAloneMode(  
    boolean bStandAloneMode  
)
```

Description:

Sets the value of the `standalonemode` property.

Example:

```
eop.setStandAloneMode(false);
```

setStyleSheet

Syntax:

```
void setStyleSheet(  
    string strStyles  
)
```

Description:

Loads styles from `strStyles` into the editor. These styles will overwrite any existing styles within the editor. If you want to add styles without overwriting the existing styles use `addStyleSheet`.

Example:

```
eop.setStyleSheet("p { color:blue; }");
```

setStyleSheetURL

Syntax:

```
void setStyleSheetURL(  
    string strURL  
)
```

Description:

Loads styles from a style sheet document residing at `strURL` into the editor.

Example:

```
eop.setStyleSheetURL("http://myserver.com/styles.css");
```

setTemplate

Syntax:

```
void setTemplate(
    string strTemplate
)
```

Description:

Loads the template `strTemplate` into the editor.

Example:

```
strTemplate = " <!DOCTYPE html SYSTEM 'xhtml1-transitional-eop.dtd'>
<html>
<head>
    <title>My Template</title>
</head>
<body>
    <p>&nbsp;</p>
</body>
</html>";
eop.setTemplate(strTemplate);
```

setTemplateURL

Syntax:

```
void setTemplateURL(
    string strTemplateURL
)
```

Description:

Loads a template from the URL `strTemplateURL` into the editor.

Example:

```
eop.setTemplateURL("http://myserver.com/template.xhtml");
```

setWebDAVCookieString

Syntax:

```
void setWebDAVCookieString(
    String cookie
);
```

Description:

Sets a cookie to be used in all of the editor's WebDAV connections. The string should consist of name=value pairs separated by a semicolon: "name1=value1;name2=value2;...".

Example:

```
eop.setWebDAVCookieString("myWebdavRepository=WebdavPort/Path;JSESSION=A4545B6767CBEA345");
```

showWYSIWYGAreaOnly

Syntax:

```
void showWYSIWYGAreaOnly(
    boolean bShow
)
```

Description:

If `bShow` is true, only the editor's WYSIWYG area will be visible. The whole canvas, UI and tab pane are hidden and only the WYSIWYG area is displayed. Setting `bShow` to false reverses this effect.

Example:

```
eop.showWYSIWYGAreaOnly(true);
```

uploadImages

Syntax:

```
void uploadImages(
    string strFunc
)
```

Description:

Calling this function will open the "Upload Images" dialog. When the "OK" or "Cancel" button is clicked in this dialog, the applet will call the JavaScript function `strFunc` and pass an associative array containing information about the upload of each image that was uploaded as argument to the JavaScript function if "OK" was clicked. The JavaScript function `strFunc` will receive `false` as argument if "Cancel" was clicked.

Example:

```
eop.uploadImages("saveFunc");
```

Note:

Calling the `uploadImages` function inside an event queue will halt the event cycle. I.e., if you are calling any other methods that need to be registered in the queue after the `uploadImages` function, call `pumpEvents` within the definition of `uploadImages` ' event handler.

Important:

This method is deprecated as of edit-on Pro.2. Please use `uploadToRepository` instead.

uploadToRepository

Syntax:

```
void uploadToRepository(
    string strType
    string strFunc
)
```

Description:

Calling this function will open the upload dialogs of the type specified by `strType`. The possible types are `UPLOADALL`, `UPLOADOBJECTSANDIMAGES`, `UPLOADIMAGES`, `UPLOADOBJECTS` and `UPLOADDOCUMENT`. When the "OK" or "Cancel" button is clicked in this dialog, the applet will call the JavaScript function `strFunc` and pass an associative array containing information about the upload of each file that was uploaded as argument to the JavaScript function if "OK" was clicked. The JavaScript function `strFunc` will receive `false` as argument if "Cancel" was clicked.

Example:

```
eop.uploadToRepository("UPLOADIMAGES", "saveFunc");
```

Note:

Calling the `uploadToRepository` function inside an event queue will halt the event cycle. I.e., if you are calling any other methods that need to be registered in the queue after the `uploadToRepository` function, call `pumpEvents` within the definition of `uploadToRepository` ' event handler.

resizeEditor

Syntax:

```
void resizeEditor(
    int width
    int height
)
```

Description:

Resizes the editor to the specified `width` and `height` in pixels.

Example:

```
eop.resizeEditor(600, 400);
```

showEditor

Syntax:

```
void showEditor()
```

Description:

Displays the editor if it was previously hidden using `hideEditor` or loaded using `setHiddenLoading` with the argument `displayOnLoad` set to `false`.

Although this is method is a set method, it is exempt from the queue. Thus, the method is executed immediately when it is called, and no calling of `pumpEvents` is required. This method does not tamper with the event queue in any way.

Example:

```
eop.showEditor();
```

hideEditor

Syntax:

```
void hideEditor(
    boolean noCollapse
)
```

Although this method is a set method, it is exempt from the queue. Thus, the method is executed immediately when it is called, and no calling of `pumpEvents` is required. This method does not tamper with the event queue in any way.

Description:

Hides the editor when it is called. If `noCollapse` is set to true, the editor will behave like an XML element with the style "visibility: hidden", i.e. while it will no longer be visible, it will still take up the same space as before it was hidden.

If `noCollapse` is set to false, the editor is hidden and takes up a minimal amount of space in the document while it is hidden.

Example:

```
eop.showEditor();
```

Although this method is a set method, it is exempt from the queue. Thus, the method is executed immediately when it is called, and no calling of `pumpEvents` is required. This method does not tamper with the event queue in any way.

3.3 User Interface Configuration

3.3.1 Structure

uiconfig

Description:

Wrapper for the elements `actions`, `menubar` and `contextmenu` which in turn group the other elements of the user interface configuration file.

Parent:

None.

Children:

`actions`, `menubar`, `contextmenu`

Attributes:

`iconstyle` (classic|modern) #IMPLIED

Determines the icon set that is used in the editor. There are two icon sets: classic and modern. The default is "classic".

Example:

```
<uiconfig iconstyle="modern"> ... </uiconfig>
```

3.3.1.1 Elements Defining Actions

actions

Description:

Wrapper for the elements `action`, `actionurl` and `actionjs` which allow the definition of custom actions or the overwriting of default actions.

Parent:

`uiconfig`

Children:

`action`, `actionurl`, `actionjs`

Attributes:

`defaultshortcuts` (enabled|disabled) "enabled"

If `defaultshortcuts` is set to "disabled", the default keyboard shortcuts of all editor actions are disabled.

Example:

```
<actions> <action id="REDO" /> </actions>
```

action**Description:**

Allows to define a custom action or to overwrite a default action.

Parent:

actions

Children:

shortcut

Attributes:

id CDATA #REQUIRED

Determines the unique id of the action. This id will be used by a toolbar button, pull down menu item or context menu item to refer to this action.

icon CDATA #IMPLIED

Indicates the location of an icon which will be used to display this action in the pull down menu, the toolbar and the context menu. The path may be relative to the editor's codebase.

class CDATA #IMPLIED

Indicates a custom Java class to be called when this action is fired.

enabledinsourceview (true|false) "false"

Determines whether the action will be enabled in source view or not.

Note:

The default actions defined in edit-on Pro may have no effect in source view. The actions available per default were implemented with attention to the WYSIWYG mode only.

enabledinreadonlymode (true|false) "false"

Determines whether the action will be enabled when the editor is in read-only mode view or not.

enabled (true|false) "true"

Determines whether the action is enabled.

defaultshortcut (enabled|disabled) "enabled"

If `defaultshortcut` is set to "disabled", the default keyboard shortcuts of the corresponding editor action are disabled.

alwaysenabledinwysiwyg (true|false) "false"

Determines whether the action is always enabled in WYSIWYG mode. If set to true, the action is even active when the caret is placed within read-only elements, or if the editor is in read-only mode.

Example:

```
<action id="REDO" icon="icons/templatepicker.gif"
  class="com.realobjects.customdialogs.templatepicker.TemplatePicker"
  enabledinsourceview="false">
  <shortcut id="control Z" default="true" />
</action>
```

actionjs

Description:

Allows to define a custom JavaScript action.

Parent:

actions

Children:

shortcut, parameterjs

Attributes:

id CDATA #REQUIRED

Determines the unique id of the action. This id may be used by a tool bar button, pull down menu item or context menu item to refer to this action.

icon CDATA #IMPLIED

Indicates the location of an icon which will be used to display this action in the pull down menu, the toolbar or the context menu. The path may be relative to the editor's codebase.

jsfunction CDATA #REQUIRED

The name of a JavaScript function to be called when this action is fired. The function must be defined in the web page containing the editor. The specified function will receive a reference to the editor as first parameter to facilitate the communication with the editor.

enabledinsourceview (true|false) "false"please

Determines whether the action is enabled in source view or not.

enabled (true|false) "true"

Determines whether the action is enabled.

Example:

```
<actionjs id="INSERTDATE" jsfunction="insertDate">
  <shortcut id="control ." default="true" />
</actionjs>
```

parameterjs

Description:

Allows to define a parameter that will be received by the JavaScript function specified in its parent jsaction element.

Parent:

actions

Children:

shortcut, parameterjs

Attributes:

parameter CDATA #REQUIRED

Specifies a parameter that will be received by the JavaScript function specified in its parent jsaction element. The first parameter received by the JavaScript function will always be a reference to the editor, this means additional parameters passed via parameterjs will be passed as second, third, etc. parameters.

Example:

```
<actionjs id="INSERTDATE" jsfunction="insertDate">
  <parameterjs parameter="2004-09-16" />
</actionjs>
```

actionurl

Description:

Allows to define an action opening an URL when fired.

Parent:

actions

Children:

shortcut

Attributes:

id CDATA #REQUIRED

Determines the unique id of the action. This id may be used by a toolbar button, pull down menu item or context menu item to refer to this action.

icon CDATA #IMPLIED

Indicates the location of an icon which will be used to display this action in the pull down menu, the toolbar or the context menu. The path may be relative to the editor's codebase.

target CDATA #IMPLIED

Determines the target frame in which the URL will be opened.

url CDATA #REQUIRED

Specifies the URL that will be opened when the action is fired.

enabledinsourceview (true|false) "false"

Determines whether the action is enabled in source view or not.

enabled (true|false) "true"

Determines whether the action is enabled at all.

Example:

```
<actionurl id="ROWEBSITE"
  url="http://www.realobjects.com"
  target="_new" />
```

shortcut

Description:

Allows to define a shortcuts for an action. The shortcuts will fire the action when they are used.

Parent:

action, actionjs, actionurl

Children:

None.

Attributes:

id CDATA #REQUIRED

Specifies a keystroke combination for the shortcut.

default (true|false) "false"

If multiple shortcuts are defined for an action, only the default shortcut will be displayed in the editor menus.

Example:

```
<shortcut id="control Z" default="true" />
```

dragndropaction

Description:

Allows to specify the action that will handle drag & drop events.

Parent:, please

actions

Attributes:

actionid CDATA #REQUIRED

Specifies the action that will handle drag & drop events.

Example:

```
<dragndropaction actionid="DRAGNDROPWITHFILTER" />
```

3.3.1.2 Elements Defining the Pull Down Menu and Context Menu

menubar

Description:

Wrapper for the `menu` elements which define the individual menus available in the pulldown menu's menubar.

Parent:

uiconfig

Children:

menu

Attributes:

None.

Example:

```
<menubar>
  <menu localeid="&File">
    <menuitem actionid="NEW" />
  </menu>
  <menu localeid="&Edit">
    <menuitem actionid="COPY"/>
  </menu>
</menubar>
```

menu

Description:

Defines a menu in either the context menu or the pull down menu. A menu may contain other menus as well as separators and menu items.

Parent:

menubar, menu, contextmenu

Children:

menu, menuitem, separator

Attributes:

localeid CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the menu depending on the locale is used.

Note:

"&" can be used within `localeid` to specify a mnemonic for the corresponding menu or menuitem.

Example:

```
<menu localeid="&File">
  <menuitem actionid="NEW"/>
  <menuitem actionid="LOAD"/>
  <menuitem actionid="OPENFILE"/>
  <separator/>
  <menuitem actionid="DOCSTAT"/>
</menu>
```

menuitem

Description:

Defines a menuitem inside a menu.

Parent:

menu

Children:

None.

Attributes:

localeid CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the menu item depending on the locale that is used.

actionid CDATA #REQUIRED

Determines the id of the action that is fired when the menuitem is clicked.

Example:

```
<menuitem localeid="&New" actionid="NEW"/>
```

separator

Description:

Defines a separator inside a menu or the toolbar.

Parent:

menu | toolbar

Children:

None.

Attributes:

None.

Example:

```
<separator />
```

3.3.1.3 Elements Defining The Toolbar

toolbar

Description:

Defines the editor's toolbar.

Parent:

uiconfig

Children:

button, buttongroup, choice, separator, endrow

Attributes:

collapse (true|false) "false"

Specifies whether the toolbar is collapsed to one line after the editor is initialized. The toolbar may be expanded manually.

classicbuttonstyle (true|false)

Specifies whether toolbar buttons are painted by Swing or by edit-on Pro. If set to true, the buttons are painted by Swing.

Example:

```
<toolbar collapse="false"
  classicbuttonstyle="false">
  <button localeid="New" actionid="NEW" enabled="true"/>
  <button localeid="Open" actionid="LOAD"/>
  <separator/>
  <button localeid="Cut" actionid="CUT"/>
</toolbar>
```

button

Description:

Defines a toolbar button.

Parent:

toolbar

Children:

None.

Attributes:

localeid CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the button name depending on the locale that is used.

actionid CDATA #REQUIRED, please

Determines the id of the action that is fired when the menuitem is clicked.

Example:

```
<button actionid="CUT" />
```

buttongroup

Description:

Specifies a special button type which functions like a normal button element. Additionally, the `buttongroup` has a small arrow selector allowing the user to select one out of multiple `subbutton` buttons.

Parent:

toolbar

Children:

subbutton

Attributes:

localeid CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the button name depending on the locale that is used.

actionid CDATA #REQUIRED

Determines the id of the action that is fired when the menuitem is clicked.

Example:

```
<buttongroup localeid="Ordered List" actionid="ORDEREDLIST">
  <subbutton localeid="Lower Alpha Orderedlist" actionid="LOWERALPHAORDEREDLIST"/>
  <subbutton localeid="Upper Alpha Orderedlist" actionid="UPPERALPHAORDEREDLIST"/>
  <subbutton localeid="Number Alpha Orderedlist" actionid="NUMBERALPHAORDEREDLIST"/>
</buttongroup>
```

subbutton

Description:

Specifies a button being displayed after clicking on the arrow selector of the parent `buttongroup`.

Parent:

`buttongroup`

Children:

None.

Attributes:

`localeid` CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the button name depending on the locale used.

`actionid` (DISCUNORDEREDLIST | CIRCLEUNORDEREDLIST | SQUAREUNORDEREDLIST | UPPERALPHAORDEREDLIST | LOWERALPHAORDEREDLIST | NUMERICORDEREDLIST | LOWERROMANORDEREDLIST | UPPERROMANORDEREDLIST) #REQUIRED

Determines the id of the action fired when the menuitem is clicked. Only ordered lists or unordered lists can be `subbutton` elements.

`enabled` (true|false) "true"

Determines whether the button is enabled.

Example:

```
<subbutton localeid="Number Alpha Orderedlist"
  actionid="NUMERICORDEREDLIST" />
```

choice

Description:

Defines a toolbar combo box.

Parent:

`toolbar`

Children:

None.

Attributes:

`localeid` CDATA #IMPLIED

The editor will search for this `localeid` in the locale specified by the `locale` or `localeurl` property and display the corresponding `locale` string. If no corresponding `localeid` can be found, the `localeid` is displayed textually. If this attribute is not specified, the editor will display a default localized version of the combo box name depending on the locale that is used.

`actionid` (FONTSIZE|FONTTYPE|STYLESHEET|PARAGRAPHSTYLE) #REQUIRED

Determines the id of the action that is fired when the menuitem is clicked.

`width` CDATA #IMPLIED

Determines the width of the combo box.

`fontsize` CDATA #REQUIRED

Determines the font size that is used in the combo box.

Example:

```
<choice localeid="Paragraph Style"
  actionid="PARAGRAPHSTYLE"/>
```

customchoice

Description:

May be used to specify a custom drop down box within the editor's toolbar. The drop down box groups several default editor actions as well as custom actions.

Parent:

config

Children:

customchoicetitem

Attributes:

localeid CDATA #REQUIRED | fontsize CDATA #IMPLIED

Determines a custom drop down box. The required attribute `localeid` defines the tooltip, `fontsize` sets the font size of the elements displayed within the drop down box.

Example:

```
<customchoice localeid="Custom choice tooltip"
  fontsize="20">
```

customchoicetitem

Description:

Adds the specified editor action or custom action to the `customchoice` toolbar drop down box

Parent:

customchoice

Children:

None.

Attributes:

actionid CDATA #REQUIRED | localeid CDATA #IMPLIED

The attribute `actionid` determines the ACTIONID which should be listed in the drop down box. `localeid` changes the default label for respective ACTIONID within the drop down box.

Example:

```
<customchoicetitem actionid="BOLD" localeid="Format Bold"/>
```

endrow

Description:

May be used to specify the end of a row in the toolbar. If no `endrow` elements are used, the editor will determine automatically where to wrap the toolbar.

Parent:

toolbar

Children:

None.

Attributes:

None.

Example:, please

```
<endrow />
```

3.3.2 Actions

ACCEPTALL

Description:

Accepts all changes in the document. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

ACCEPTDELETION

Description:

Accepts a deletion at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

ACCEPTFORMAT

Description:

Accepts a formatting change at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

ACCEPTINSERTION

Description:

Accepts an insertion at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

ACCEPTALLSELECTEDCHANGES

Description:

Accepts all changes within the current selection. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

ACRONYM

Description:

Opens the "Insert Acronym" dialog.

ACRONYMPROPERTIES

Description:

Opens the "Acronym Properties" dialog.

ADVANCEDORDEREDLIST

Description:

Allows the user to select one of the available ordered list types.

ADVANCEDREDO

Description:

Proposes a list of undone actions that can be redone.

ADVANCEDSTYLESHEET

Description:

Provides a list of CSS styles to select from. The selected CSS style is applied to the current element or selection. Additionally, the list displays to which elements the individual styles can be applied to.

ADVANCEDUNDO

Description:

Proposes a list of actions that can be undone.

ADVANCEDUNORDEREDLIST

Description:

Allows the user to select one of the available unordered list types.

ALIGNLEFT

Description:

Aligns the current element or selection on the left.

ALIGNCENTER

Description:

Aligns the current element or selection at the center.

ALIGNJUSTIFY

Description:

Justifies the current element or selection.

ALIGNRIGHT

Description:

Aligns the current element or selection on the right.

AUTOSPELLCHECK

Description:

Enables / disables real-time spell-checking.

AUTOSPELLCHECKSUGGESTION

Description:

Provides a list of correction suggestions if the word at the cursor position was not recognized by the spell checker.

Note:

This action may only be used in the context menu.

BLOCKQUOTE

Description:

Opens the "Insert Blockquote" dialog.

BLOCKQUOTEPROPERTIES

Description:

Opens the "Blockquote Properties" dialog.

BOLD

Description:

Makes the selection bold and enables typing of bold text.

BOOKMARK

Description:

Opens the "Insert Bookmark" dialog.

BOOKMARKPROPERTIES

Description:

Opens the "Edit Bookmark Properties" dialog if the current element or selection is a bookmark.

BR

Description:

Inserts a line break (
) at the current caret position.

CDATAPROPERTIES

Description:

Opens the "Edit CDATA" dialog if the current element or selection is a CDATA section.

CELLPROPERTIES

Description:

Opens the "Cell Properties" dialog if the current element or selection is a table cell.

CIRCLEUNORDEREDLIST

Description:

Displays a list with each member of the list marked with a circle.

CHANGECASE

Description:

Opens the "Change case" dialog.

CLEAN

Description:

Removes formatting from the current selection.

COLOR

Description:

Opens the "Font Color" dialog.

COLUMNPROPERTIES

Description:

Opens the "Column Properties" dialog if the current element or selection is a table column.

COMMENT

Description:

Opens the "Insert Comment" dialog.

The inserted comment will only be visible in WYSIWYG mode if the "Show All" option (P-mode) is enabled.

COMMENTPROPERTIES

Description:

Opens the "Edit Comment" dialog if the current element or selection is a comment.

CONTEXTMENU

Description:

Opens the context menu.

CONVERTTABLEWIDTHTORELATIVE

Description:

Converts the table at the caret position to a table with relative width.

COPY

Description:

Copies the current selection to the clipboard.

CUT

Description:

Cuts the current selection and inserts it into the clipboard.

DECINDENT

Description:

Decreases the indentation of a paragraph or list element.

DELETECOLUMN

Description:

Deletes the table column at the cursor position.

DELETETABLE

Description:

Deletes the table at the cursor position.

DIFF

Description:

Opens the "Compare Documents" dialog which allows the user to select two documents for comparison. Once the documents are loaded, the editor runs in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

DIFFLIST

Description:

Opens the "Diff List" dialog which displays all modifications found in the document in a list.

This feature is only available if VersioTrack is enabled in your license.

DISCUNORDEREDLIST

Description:

Displays a list with each member of the list marked with a disc.

DOCSTAT

Description:

Opens the "Document Statistics" dialog.

ENDENCLOSINGBLOCK

Description:

Ends the enclosing block. That is, if the caret is within a `blockquote`, `div`, `ol` or `ul` element, a new `p` element is inserted after this element, and the caret is placed within the newly inserted `p` element.

FIND

Description:

Opens the "Find and Replace" dialog.

HIGHLIGHTTEXT

Description:

Opens a dialog which lets the user select a highlight color for the current selection and enables typing of highlighted text.

FONTSIZE

Description:

Provides a list of font sizes to select from. The selected font size is applied to the current selection.

FONTTYPE

Description:

Provides a list of font styles to select from. The selected font style is applied to the current selection.

GOTONEXTBOOKMARK

Description:

Sets the caret at the end of the next bookmark found in the document starting from the caret position.

GOTOPREVIOUSBOOKMARK

Description:

Sets the caret at the end of the previous bookmark found in the document starting from the caret position.

HARDRULERPROPERTIES

Description:

Opens the "Horizontal Line Properties" dialog if the current element or selection is a hard ruler.

HELP

Description:

Opens the online help in a new browser window.

Note:

Integrators/developers of course can modify the help files according to the individual need of their installation. Upon request RealObjects will also provide the related DocBook files and style sheets.

HR

Description:

Inserts a hard ruler at the cursor position.

IMAGE

Description:

Opens the "Insert Image" dialog.

IMAGEPROPERTIES

Description:

Opens the "Image Properties" dialog if the current element or selection is an image.

INFO

Description:

Opens the "Info" dialog.

INCIDENT

Description:

Increases the indentation of a paragraph or list element.

INLINEQUOTE

Description:

Opens the "Insert Inline Quote" dialog.

INLINEQUOTEPROPERTIES

Description:

Opens the "Inline Quote Properties" dialog.

INSERTCOLUMN

Description:

Opens the "Insert Column" dialog if the cursor is inside a table.

INSERTCUSTOMTAG

Description:

Opens the "Insert Custom Tag" dialog.

INSERTDATETIME

Description:

Opens the "Insert date and time" dialog.

INSERTLANG

Description:

Opens the "Insert Language Attribute" dialog. This dialog inserts a `span` element with a "lang" attribute around the selected content. The value of the "lang" attribute is the value specified by the dialog

INSERTROW

Description:

Opens the "Insert Row" dialog if the cursor is inside a table.

INSERTSPAN

Description:

Opens the "Insert Span" dialog.

INSERTTABLE

Description:

Inserts a default table at the cursor position.

INSERTTABLEDIALOG

Description:

Opens the "Insert Table Dialog".

INSERTTABLEWIZARD

Description:

Opens the "Insert Table Wizard".

ITALIC

Description:

Makes the selection italic and enables typing of italic text.

LANGPROPERTIES

Description:

Opens the "Edit Language Attribute" dialog if the caret is located in a `span` tag having a "lang" attribute.

LISTPROPERTIES

Description:

Opens the "List Properties" dialog.

LINK

Description:

Opens the "Insert Hyperlink" dialog.

LINKPROPERTIES

Description:

Opens the "Edit Hyperlink" dialog if the current element or selection is a hyperlink.

LOAD

Description:

Opens the "Open URL" dialog.

LOWERALPHAORDEREDLIST

Description:

Displays an ordered list with each member of the list marked with a lowercase type ('a').

LOWERROMANORDEREDLIST

Description:

Displays a list with each member of the list is marked with a lowercase Roman number ('iv').

MERGECELLABOVE

Description:

Merges the table cell at cursor position with the cell above it.

MERGECELLBELOW

Description:

Merges the table cell at cursor position with the cell below it.

MERGECELLLEFT

Description:

Merges the table cell at cursor position with the cell at the left.

MERGECELLRIGHT

Description:

Merges the table cell at cursor position with the cell at the right.

NEW

Description:

Clears the editor's content and populates the editor with a template or default document.

NEXTDIFF

Description:

Jumps to the next modification found in the document.

This feature is only available if VersioTrack is enabled in your license.

NUMERICORDEREDLIST

Description:

Displays an ordered list with each member of the list marked with an Arabic number ('1').

OBJECT

Description:

Opens the "Insert Object" dialog.

OBJECTPROPERTIES

Description:

Opens the "Image Properties" dialog if the current element or selection is an image.

OPENFILE

Description:

Opens the file chooser dialog.

ORDEREDLIST

Description:

Displays a list where each member of the list is marked with an Arabic number ('1'). Additionally allows the user to select one of the three available ordered list types.

PAGEPROPERTIES

Description:

Opens the "Page Properties" dialog, which allows to specify properties for the document currently being edited, such as the document title, the default color of hyperlinks and a background image for the document.

PARAGRAPHSTYLE

Description:

Provides a list of paragraph styles to select from. The selected paragraph style is applied to the current selection.

PASTE

Description:

Inserts the contents of the clipboard in HTML format with style information contained inline.

PASTEPLAINTEXT

Description:

Inserts the contents of the clipboard without any formatting or styles.

PASTESPECIAL

Description:

Opens the "Paste Special" dialog. This dialog lets the user select which type of paste to perform (e.g plain text only, or in HTML but without styles etc.) and will give a short overview over what each type of paste will result in.

PASTEWITHFILTER

Description:

Inserts the contents of the clipboard in HTML format, but without styles.

PASTEWITHSTYLEDEFS

Description:

Inserts the contents of the clipboard in HTML format with style information contained in an embedded style sheet, e.g.converts inline styles to style classes. Imported styles will be merged with existing styles respectively style classes.

PIPROPERTIES

Description:

Opens the "Edit Processing Instruction" dialog if the current element or selection is a processing instruction.

PREVDIFF

Description:

Jumps to the previous modification found in the document.

This feature is only available if VersioTrack is enabled in your license.

REDO

Description:

The last action that was performed is rolled back.

REJECTALL

Description:

Rejects all changes within the document. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

REJECTDELETION

Description:

Rejects a deletion at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

REJECTFORMAT

Description:

Rejects a formatting change at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

REJECTINSERTION

Description:

Rejects an insertion at the caret position. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

REJECTSELECTEDCHANGES

Description:

Rejects all changes within the current selection. This action is only enabled in comparison mode.

This feature is only available if VersioTrack is enabled in your license.

REMOVEBOOKMARK

Description:

Removes a bookmark at the cursor position.

REMOVECUSTOMTAG

Description:

Removes a custom tag at the cursor position. Only the custom tag itself is removed, while its content is preserved.

REMOVECUSTOMTAGWITHCONTENT

Description:

Removes a custom tag at the cursor position. The custom tag is removed including all of its content.

REMOVELINK

Description:

Removes a hyperlink from the current selection.

RESETIMAGESIZE

Description:

If an image is selected or the caret is placed on an image when this action is called, the image size is reset to its original size if it was modified.

RESETUSERPREFS

Description:

Resets the user preferences. The user preferences consist of the values that were last specified by the user in some dialogs such as the table properties dialog, the color properties dialog or the cell properties dialog.

ROWPROPERTIES

Description:

Opens the "Row Properties" dialog if the current element or selection is a table row.

SAVE

Description:

Posts the editor's content to the URL specified by the `gethtmldataurl` property.

SAVEAS

Description:

Opens a "Save as" dialog which allows the user to save the document currently being edited locally on his computer.

SELECTALL

Description:

Selects all content within the editor.

SHOWNALLCHAR

Description:

Displays control characters, XML comments, CDATA sections, processing instructions and custom tags.

SHOWSOURCEVIEW

Description:

Switches to `Source View` when it is activated and to `WYSIWYG View` when it is deactivated.

SHOWTABLEGRID

Description:

Enables / disables the display of the table grid. If it is enabled, table borders with a width of "0" will be displayed as a dotted line. If it is disabled, borders with no width will not be displayed.

SHOWWYSIWYGAREAONLY

Description:

Enables / disables the display of the canvas, UI and tab pane. When enabled, only the WYSIWYG area is displayed.

Note:

When this action is enabled, most of the UI is hidden. The user won't be able to use the toolbar or pull down menu to disable this action. Thus, it is recommended to provide a context menu and a keyboard shortcut for this action.

SHOWWYSIWYGVIEW

Description:

Switches to `WYSIWYG view` when it is activated and to `Source View` when it is deactivated.

SOURCESYNTAXHIGHLIGHT

Description:

Enables / disables syntax highlighting in source view when it is called.

SOURCELINESWRAP

Description:

Enables / disables word wrap in source view when it is called.

SPECIALCHARACTER

Description:

Opens the "insert Symbol" dialog.

SPELLCHECK

Description:

Checks the spelling of the content inside the editor. If the spell checker finds one or more errors, the "Check Spelling" dialog is opened.

SPLITCELL

Description:

Opens the "split" dialog, which allows splitting of table cells.

SPLITPARAGRAPH

Description:

Splits the paragraph at the caret position. This has the same effect when as pressing the "enter" key when the configuration option "alternateenterkeyaction" is not set.

SQUAREUNORDEREDLIST

Description:

Displays a list with each member of the list marked with a square.

STANDALONE

Description:

Enables / disables the "Stand Alone" mode. In "Stand Alone" mode the editor is displayed in a separate window outside of the browser and can be freely resized by the user.

STRIKETHROUGH

Description:

Makes the selection striked out and enables typing of striked out text.

STYLESHEET

Description:

Provides a list of CSS styles to select from. The selected CSS style is applied to the current element.

STYLESHEETPROPERTIES

Description:

Opens the "Style Properties" dialog.

SUBSCRIPT

Description:

Makes the selection subscript and allows the typing of subscript text.

SUPERSCRIPT

Description:

Makes the selection superscript and allows the typing of superscript text.

TABLEAUTOFORMAT

Description:

Opens the "Table Autoformat" dialog if the element at the cursor position is a table. This dialog allows you to select and apply a template to a table.

TABLEPROPERTIES

Description:

Opens the "Table Properties" dialog if the element at the cursor position is a table.

TAGPROPERTIES

Description:

Opens the "Edit Tag Properties" dialog if the current element or selection is a custom element.

THESAURUS

Description:

Opens the thesaurus for the word at the current caret position.

THESAURUSSUGGESTION

Description:

Provides a list of thesaurus suggestions for the word at the current caret position for use in the context menu.

TRACKCHANGES

Description:

Enables comparison mode in the editor. All changes made to the document currently edited are tracked from then on.

This feature is only available if VersioTrack is enabled in your license.

TREEVIEW

Description:

Opens the "Tree View" dialog. The tree view will display the position of the element at the cursor position inside the document tree.

UNDERLINE

Description:

Underlines the selection and enables typing of underlined text.

UNDO

Description:

The last action performed inside the editor is undone.

UNORDEREDLIST

Description:

Displays a list with each member of the list is marked with a disc. Additionally allows the user to select one of the three available unordered list types.

UPLOADDOCUMENTIMAGES

Description:

Opens the "Upload Document Images" dialog. The dialog allows the user to upload images to the server if the `filemanagement` element is specified in the configuration file.

Important:

This action is deprecated as of edit-on Pro version 4.2. Please use `UPLOADIMAGES` instead.

UPLOADDOCUMENT

Description:

Opens the "Upload Document" dialog. The dialog allows the user to upload the document currently being edited to the server. To configure where the document should be uploaded to, please use the `filemanagement` setting in the configuration file.

UPLOADALL

Description:

Successively opens the "Upload Objects" dialog and the "Upload Document" dialog. The dialogs resp. allows the user to upload images and objects and the document currently being edited. To configure where the images should be uploaded to, please use the `filemanagement` setting in the configuration file.

UPLOADIMAGES

Description:

Opens the "Upload Images" dialog. The dialog allows the user to upload images to the server. To configure where the images should be uploaded to, please use the `filemanagement` setting in the configuration file.

UPLOADOBJECTS

Description:

Opens the "Upload Objects" dialog. The dialog allows the user to upload objects to the server. To configure where the objects should be uploaded to, please use the `filemanagement` setting in the configuration file.

UPPERALPHAORDEREDLIST

Description:

Displays a list with each member of the list is marked with an uppercase type ('A').

UPPERROMANORDEREDLIST

Description:

Displays a list with each member of the list is marked with an uppercase Roman number ('IV').

3.4 CSS Reference

3.4.1 Style Sheets Supported by edit-on Pro

edit-on Pro supports a subset of the CSS properties. Properties not listed here are preserved by the editor, but will not be rendered.

3.4.1.1 Font properties

Font family

Syntax:

```
font-family:font1,font2,fontn;
```

Possible values:

Any font name.

Example:

```
font-family:arial,verdana;
```

Font size

Syntax:

```
font-size:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

```
xx-small | x-small | small | medium | large | x-large | xx-large
```

Example:

```
font-size:12px;
```

Font style

Syntax:

```
font-style:value;
```

Possible values:

```
normal | italic
```

Example:

```
font-style:italic;
```

Font weight

Syntax:

```
font-weight:value;
```

Possible values:

```
normal | bold
```

Example:

```
font-weight:bold;
```

3.4.1.2 Color and background properties

Background

Syntax:

```
background: value;
```

Possible values:

```
color | transparent
```

Determine a `color` value using one of the following syntaxes: `#0000ff`, `black`, `rgb(50%,60%,80%)`

Example:

```
background:#0000ff;
```

Background attachment

Syntax:

```
background-attachment:value;
```

Possible values:

```
fixed | scroll
```

Determine the scrolling behaviour of a background image.

Example:

```
background-image:url(background.jpg);
background-repeat:repeat-y;
background-attachment:fixed;
```

Background color

Syntax:

```
background-color:value;
```

Possible values:

```
color | transparent
```

Determine a `color` value using one of the following syntaxes: `#0000ff`, `black`, `rgb(50%,60%,80%)`

Example:

```
background-color:#0000ff;
```

Background image

Syntax:

```
background-image:url(URI);
```

Possible values:

```
url(URI)
```

Determine an absolute or relative URL value for the image file (GIF / JPG / JPEG / PNG) which should be display in the background.

Example:

```
background-image:
url(http://myserver.com/images/background.gif);
```

Background repeat

Syntax:

```
background-repeat:value;
```

Possible values:

```
repeat | repeat-x | repeat-y | no-repeat
```

Determine the repetition of an image (set by `background-image`) in background.

Example:

```
background-image:url(background.jpg);
background-repeat:repeat-y;
```

Color

Syntax:

```
background:value;
```

Possible values:

```
color
```

Determine a `color` value using one of the following syntaxes: `#0000ff`, `black`, `rgb(50%,60%,80%)`

Example:

```
color:#0000ff;
```

3.4.1.3 Text properties

Text alignment

Syntax:

```
text-align:value;
```

Possible values:

```
left | right | center
```

Example:

```
text-align:center;
```

Text decoration

Syntax:

```
text-decoration:value;
```

Possible values:

```
none | underline | line-through
```

Example:

```
text-decoration:line-through;
```

Vertical alignment

Syntax:

```
vertical-align:value;
```

Possible values:

```
sub | super | baseline
```

Example:

```
vertical-align:sub;
```

3.4.1.4 Box properties

Bottom margin

Syntax:

```
margin-bottom:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

Example:

```
margin-bottom:12pt;
```

Note:

This style property will only be rendered for the body element.

Top margin

Syntax:

```
margin-top:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

Example:

```
margin-top:12pt;
```

Note:

This style property will only be rendered for the body element.

Left margin

Syntax:

```
margin-left:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

Example:

```
margin-left:12pt;
```

Right margin

Syntax:

```
margin-right:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

Example:

```
margin-right:12pt;
```

3.4.1.5 List properties

List style

Syntax:

```
list-style-type:value;
```

Possible values:

```
disc | circle | square | decimal | lower-alpha | uppper-alpha
```

Example:

```
list-style-type: disc;
```

List style image

Syntax:

```
list-style-image: url(value);
```

Example:

```
ul { list-style-type:none; list-style-image:url(icon.gif); }
```

3.4.1.6 Table properties

Table width

Syntax:

```
width:value;
```

Possible values:

--px | --%

Example:

width:100%;

Border color**Syntax:**

border-color:value;

Possible values:

color

Determine a `color` value using one of the following syntaxes: `#0000ff`, `black`, `rgb(50%,60%,80%)`**Example:**

border-color:#0000ff;

Border width**Syntax:**

border-width:value;

Possible values:

--px

Example:

border-width:2px;

3.4.2 edit-on Pro CSS Extensions**Element action****Syntax:**

ro-action:value;

Possible values:

Any action defined in the editor's configuration file.

Description:The action `value` will be available in the context menu for the elements this style applies to.**Note:**The action `value` will only be available for those elements if it is added to the context menu definition in the configuration file.**Example:**

```
p { ro-action: MYCUSTOMACTION; }
body { ro-action: CUSTOMACTION1, CUSTOMACTION2; }
```

CDATA action**Syntax:**

ro-cdata

Description:The `ro-cdata` selector allows you to define style properties for CDATA sections.**Example:**

```
ro-cdata { ro-action:CDATAPROPERTIES; }
```

Processing instruction action

Syntax:

```
ro-pi
```

Description:

The `ro-pi` selector allows you to define style properties for processing instructions.

Example:

```
ro-pi { ro-action:PIPROPERTIES; }
```

Comment action

Syntax:

```
ro-comment:value;
```

Description:

The `ro-comment` selector allows you to define style properties for comments.

Example:

```
ro-comment { ro-action:COMMENTPROPERTIES; }
```

3.4.2.1 Style Properties Controlling the Behaviour of Custom Tags

Read-only content

Syntax:

```
ro-readonlycontent:value;
```

Possible values:

```
true|false
```

Description:

The content of the elements this style is applied to will become read-only, i.e. it can not be edited or modified. The corresponding elements still can be deleted.

Example:

```
ro-readonlycontent:true;
```

Read-only element

Syntax:

```
ro-readonlyelement:value;
```

Possible values:

```
true|false
```

Description:

The elements this style is applied to will become read-only, i.e. they can not be edited or modified.

Example:

```
ro-readonlyelement:true;
```

Content Limit

Syntax:

```
ro-contentlimit:value;
```

Possible values:

Any integer value.

Description:

This element can only contain up to `value` characters. Entering more characters than specified by `value` in this element is not possible.

Example:

```
ro-contentlimit:100;
```

Read-only element**Syntax:**

```
ro-readonlyelement:value;
```

Possible values:

```
true|false
```

Description:

The elements this style is applied to will become read-only, i.e. they can not be edited or modified.

Example:

```
ro-readonlyelement:true;
```

Formatting Only**Syntax:**

```
ro-formattingonly:value;
```

Possible values:

```
(true|false)
```

Description:

The content of this element can not be modified as with `ro-readonlycontent`. The formatting of the element however can still be modified when this property is set.

This property will not have an effect if `ro-readonlycontent` is set to true for the same element.

Example:

```
mytag {
  ro-readonlycontent:false;
  ro-formattingonly:true;
}
```

3.4.2.2 Style Properties Affecting the Display of Elements**Always show element****Syntax:**

```
ro-alwaysshowtags:value;
```

Possible values:

```
true|false
```

Description:

The custom element this style is applied to is always displayed. Normally custom elements are only displayed while the "SHOWNALLCHAR" mode is active.

Example:

```
ro-alwaysshowtags:true;
```

Start tag icon

Syntax:

```
ro-starttagicon:value;
```

Possible values:

URL pointing to the location of an icon file (possible formats: GIF, JPG and PNG). May be relative to the editor's codebase.

Description:

The editor will use the icon indicated by `value` to display the opening tag of the elements this style is applied to.

Note:

The element is only displayed if the "SHOWNALLCHAR" mode is active or if the style `ro-alwaysshowtags` is applied to this element.

Example:

```
ro-starttagicon:"icons/start.gif";
```

End tag icon

Syntax:

```
ro-endtagicon:value;
```

Possible values:

URL pointing to the location of an icon file (possible formats: GIF, JPG and PNG). May be relative to the editor's codebase.

Description:

The editor will use the icon indicated by `value` to display the closing tag of the element this style is applied to.

Note:

The element is only displayed if the "SHOWNALLCHAR" mode is active or if the style `ro-alwaysshowtags` is applied to this element.

Example:

```
ro-endtagicon:"icons/end.gif";
```

Maximum width of custom element representation

Syntax:

```
ro-maxiconwidth:value;
```

Possible values:

```
--pt | --pc | --px | --in | --cm | --mm
```

Description:

The editor will use this value to determine the maximum width of the generated custom element representation this style is applied to.

Note:

The style is only taken into consideration if no custom icon is specified for the custom element using `ro-starttagicon` and `ro-endtagicon`.

Example:

```
ro-maxiconwidth:100px;
```

Start tag label

Syntax:

```
ro-starttaglabel:value;
```

Description:

The editor will use the value indicated as label for the opening of the custom element this style is applied to. You can use the keyword `attr` to select an attribute the style applies to. The value of this attribute will then be used as label for the tag. You can freely combine attribute values and text strings in this way.

Important:

The style is only taken into consideration if no custom icon is specified for the custom element using `ro-starttagicon` and `ro-endtagicon`.

Example:

```
ro-starttaglabel:"start label" attr(nameofanattribute);
```

End tag label

Syntax:

```
ro-endtaglabel:value;
```

Description:

The editor will use the value indicated as label for the closing of the custom element this style is applied to. You can use the keyword `attr` to select an attribute the style applies to. The value of this attribute will then be used as label for the tag. You can freely combine attribute values and text strings in this way.

Important:

The style is only taken into consideration if no custom icon is specified for the custom element using `ro-starttagicon` and `ro-endtagicon`.

Example:

```
ro-endtaglabel:"end label: " attr(nameofanattribute);
```

3.4.2.3 Style Properties Controlling the Display of CSS Classes in Editor Dialogs

Hide Class

Syntax:

```
ro-hideclass:value;
```

Possible values:

```
true|false
```

Description:

If this property is set to false, it is not possible to select or apply the CSS class it is defined for in any of the editor's dialogs.

Example:

```
.redarea {
  ro-hideclass: true;
  color: red;
}
```

Class Description

Syntax:

```
ro-classdescription:value;
```

Description:

The value of this property determines the name which will be used to display the class it is defined for in the editor's dialogs. For example, if it is set to "Red Text", then the string "Red Text" will be displayed in dialogs such as the style selector combo box instead of the class name itself.

Example:

```
.redarea {
    ro-classdescription: "Red Text";
    color: red;
}
```

3.4.2.4 Style Properties Controlling the Display in Comparison Mode

Balloon Width

Syntax:

```
ro-balloon-width:value;
```

Description:

The value of this property determines the width of the balloons that display changes in comparison mode.

Example:

```
* {
    ro-balloon-width: 200px;
}
```

Balloon Color

Syntax:

```
ro-balloon-color:value;
```

Description:

The value of this property determines the color of the balloons that display changes in comparison mode.

Example:

```
* {
    ro-balloon-color: #0689cc;
}
```

Selected Balloon Color

Syntax:

```
ro-balloonselected-width:value;
```

Description:

The value of this property determines the width of selected balloons in comparison mode.

Example:

```
* {
    ro-balloonselected-width: #0689cc;
}
```

Selected Balloon Color

Syntax:

```
ro-balloonselected-color:value;
```

Description:

The value of this property determines the color of selected balloons in comparison mode.

Example:

```
* {
    ro-balloonselected-color: #0689cc;
}
```

3.5 Clean-up Class and Form Fields

3.5.1 Methods and Properties of the Client Side Clean-Up Base Class

3.5.1.1 Base class

CleanUpProcess

Description:

The class you should extend when implementing a custom clean-up process. You can override each of the four clean-up methods individually.

Example:

```
public class myCleanUp extends CleanUpProcess {
    public String cleanUpOnLoad(String strRawData) throws Exception {
        //TODO:
        implement your clean-up process
    }
}
```

3.5.1.2 Methods

cleanUpOnLoad

Syntax:

```
public String cleanUpOnLoad(
    String strRawData
)
```

Description:

Cleans up the string specified by strRawData and returns the cleaned data if no exception occurs. This method should be overridden for a custom implementation of the clean-up occurring when content is loaded.

cleanUpClipboard

Syntax:

```
public String cleanUpClipboard(
    String strRawData
)
```

Description:

Cleans up the string specified by `strRawData` and returns the cleaned data if no exception occurs. This method should be overridden for a custom implementation of the clean-up occurring when content is pasted from the clipboard.

cleanUpClipboardWithFilter

Syntax:

```
public String cleanUpClipboardWithFilter(
    String strRawData
)
```

Description:

Cleans up the string specified by `strRawData` and returns the cleaned data if no exception occurs. This method should be overridden for a custom implementation of the clean-up occurring when the `PASTEWITHFILTER` action is fired. See the chapter [Actions](#) .

cleanUpOnSave

Syntax:

```
public String cleanUpOnSave(
    String strRawData
)
```

Description:

Cleans up the string specified by `strRawData` and returns the cleaned data if no exception occurs. This method should be overridden for a custom implementation of the clean-up occurring when content is exported.

3.5.1.3 Properties

customTags

Declaration:

```
java.util.Properties customTags;
```

Description:

Contains the custom elements as a set of properties.

Example:

```
String myCustomTag = customTags.getProperty("mytag");
```

encoding

Declaration:

```
String encoding;
```

Description:

Contains the current encoding used by the applet. If no encoding was set, this field has the value `null` .

Example:

```
String curEncoding = encoding;
```

dtd

Declaration:

```
com.wutka.dtd.DTD dtd;
```

Description:

Contains the DTD currently used in the editor.

Example:

```
Object[] dtdItems = dtd.getItems();
```

dropIllegalAttributes**Declaration:**

```
boolean dropIllegalAttributes;
```

Description:

Determines whether attributes not allowed according to the current DTD will be removed.

Example:

```
if(dropIllegalAttributes) { // execute your code here }
```

dropIllegalTags**Declaration:**

```
boolean dropIllegalTags;
```

Description:

Determines whether tags not allowed according to the current DTD will be removed.

Example:

```
if(dropIllegalTags) { // execute your code here }
```

apiObject**Declaration:**

```
Object apiObject;
```

Description:

Contains a reference to the applet. Use this reference to access the applet's API methods described in the chapter [JavaScript API / Java API](#). To access these API methods within the applet, you have to cast `apiObject` to `com.realobjects.eop.japplet.SwingEditorApplet` before you can access the methods.

Example:

```
SwingEditorApplet theApplet = (SwingEditorApplet)apiObject;
theApplet.clear();
```

3.5.2 Form Fields Sent to the Server by the Server Side Clean-Up Process**Note:**

All fields sent to the server side clean-up process are form fields. The value of the form fields is of type `string`. The form fields are sent by a POST request.

HTMLDATA

Description:

Contains the data to be cleaned by the clean-up process.

TYPE

Possible values:

`cleanUpOnLoad` | `cleanUpOnSave` | `cleanUpClipboard` | `cleanUpClipboardWithFilter`

Description:

Contains the type of clean-up process to be performed. Use this to differentiate the different types of clean-up processes.

ENCODING

Description:

Contains the encoding of the document in the editor.

DROPILLEGALTAGS

Possible values:

`true` | `false`

Description:

Contains the value of the `dropillegaltags` property. If this equals "true", all tags not allowed according to the DTD specified in the editor will be removed.

DROPILLEGALATTRIBUTES

Possible values:

`true` | `false`

Description:

Contains the value of the `dropillegalattributes` property. If this equals "true", all attributes not allowed according to the DTD specified in the editor will be removed.

3.6 Keys Available in EOPDATA for each Action

Action: TAGPROPERTIES

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: LINK

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: LINKPROPERTIES

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: IMAGE

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: IMAGEPROPERTIES

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: INSERTCUSTOMTAG

allowedattributes

A list of the allowed attributes for the current element.

attributes:

A hashtable containing name / value pairs of the attributes the element currently has.

Action: INSERTTABLE

allowedattributes

A list of the allowed attributes for the current element.

tableattributes:

A hashtable containing name / value pairs of the attributes the table currently has.

cellattributes:

A hashtable containing name / value pairs of the attributes the cell currently has.

caption:

A string containing the current table caption.

captionalign:

A string containing the alignment of the table caption.

Action: TABLEPROPERTIES

allowedattributes

A list of the allowed attributes for the current element.

tableattributes:

A hashtable containing name / value pairs of the attributes the table currently has.

cellattributes:

A hashtable containing name / value pairs of the attributes the cell currently has.

caption:

A string containing the current table caption.

captionalign:

A string containing the alignment of the table caption.

Action: **CELLPROPERTIES**

allowedattributes

A list of the allowed attributes for the current element.

tableattributes:

A hashtable containing name / value pairs of the attributes the table currently has.

cellattributes:

A hashtable containing name / value pairs of the attributes the element currently has.