WebTextEditor User Interface

This whitepaper describes the user interface concept and features introduced in WebTextEditor.

Table of Contents

Overview 3
Layout 3
Sleek Design and Elegant User Interface 3
Fluid and Fixed Layout Support in Various Browsers 4
Lightweight UI Elements 4
Flexible Size in Limited Space 5
Professional Built-in Themes 8
Allow Style Merging 9
Load Custom CSS File 9
ToolBar 11
Fluid, Smart ToolBar 11
Intuitive Word 2007-style Floating ToolBar 11
ToolBar Mode 12
ToolBar Settings 13
Custom ToolBar 13
XML Format for Custom ToolBar 14
Customize toolbar programmatically from server-side 16
Perform Document Properties Actions 17
HTML ToolBar 17
View-based ToolBar 19
View 19
Rich Design View 19
Adaptive HTML View 19
Intuitive Split View 20
Preview 21
Task Pane 21

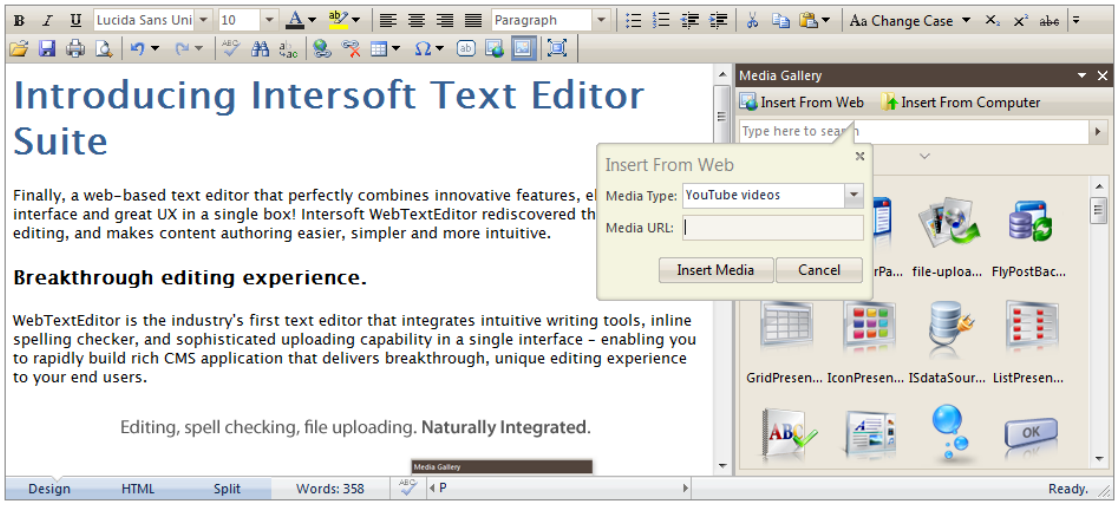
Innovative Task Pane 21
Customizable Task Pane Position 22
Media Gallery Task Pane 23
Media Gallery Settings 25
Media Resource Concept 25
Table Designer Task Pane 27
Form Control Task Pane 28
View-Based Task Pane 29
Custom Task Pane 29
Show and Hide Task Pane Programmatically 30
Context Menu 31
Enable Context Menu 31
Customize Context Menu 32
Integration with Intersoft Product 32
Integration with WebSpellChecker 32
Sophisticated Navigation Experience 32
Integration with WebFileUploader 33
Data Binding 35
Mail Merge 35
Labels 35
Preview Mail Merge 37
Send Mail 38
Mail Merge-related Server-side Event 38

Overview

WebTextEditor is the industry's first text editor that integrates intuitive writing tools, inline spelling checker, and sophisticated uploading capability in a single interface – enabling you to rapidly build rich CMS application that delivers breakthrough, unique editing experience to your end users.

WebTextEditor includes innovative features to simplify content authoring, such as:

- Reliable formatting, high-performance editor with strict XHTML compliance output.
- Sophisticated, elegant user interfaces.
- Fluid smart toolbars and Word-style floating toolbar.
- Spell check right within the editor; highlight errors in red-wave underlines and context menu for word suggestion.
- One-click automatic hints suggestion in the status bar.
- Integrated file uploader, lets users upload multiple files right within the editor.
- Sophisticated asynchronous uploading technology, enables editing while uploading is in progress.
- And more.



Intersoft WebTextEditor delivers all expected editing features, plus state-of-the-art interfaces, rich user experiences and dozens of innovative features.

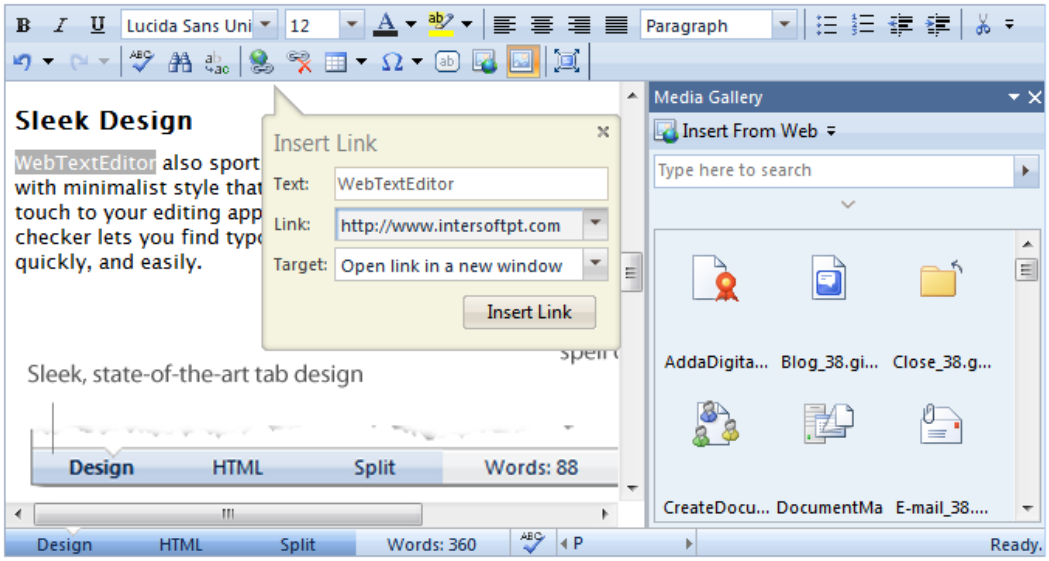
Layout

Sleek Design and Elegant User Interface

All elements in WebTextEditor have been carefully designed to be lightweight and minimalist style, yet providing powerful editing experience. The main header contains editor toolbar, which can seamlessly

adjust itself to the container size and easily hidden when user needs more space for editing. The main footer can be used for view switching, navigating through layout structure, and status bar.

Most text editor components in the market still used old-fashioned dialog box interface for user input, such as input for images, hyperlink, tables, etc – which is less intuitive because of the "blocking" nature of the dialog box interface. WebTextEditor advances editing experience with a better, more intuitive approach. Called "block-less" user interface concept, WebTextEditor supports modern, lightweight callout and task pane for user interactions, which ultimately provides user with more intuitive way to work and interact with input fields and objects.



Fluid and Fixed Layout Support in Various Browsers

WebTextEditor's layout and design have been carefully designed to support XHTML and HTML DOCTYPE and will render correctly in top browsers available in the market: IE 7, IE 8, Firefox Mozilla 3.0, Opera 9.5+, Safari 4 Beta, and Google Chrome.

By default, WebTextEditor is set to a fixed 500x700px size. When its size is set to percentage width and height, the size of WebTextEditor will be adjusted to its container size. When the container of WebTextEditor is resized, the elements of WebTextEditor will be adjusted as well. This behavior has been extensively tested to provide pixel perfect rendering in top browsers.

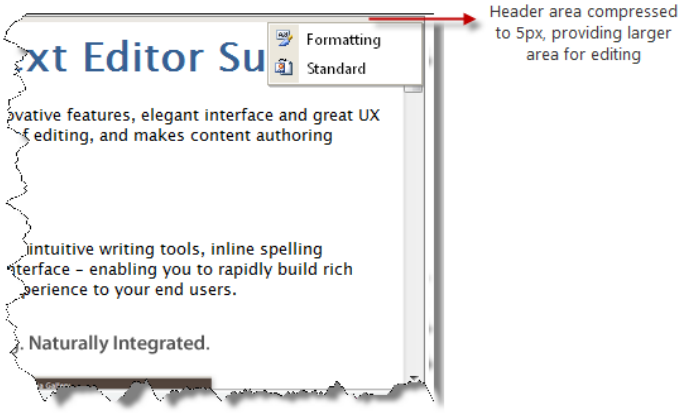
Lightweight UI Elements

A number of controls, such as toolbar, combo, color picker, table pop up, symbol, and many more are used in WebTextEditor. These controls are developed as client controls, thus providing very lightweight elements used in WebTextEditor. As client controls, they also provide high performance and responsive user interface while significantly minimizing server-side page output.

Flexible Size in Limited Space

In certain scenarios, page might contain complex layout, leaving only a limited space where WebTextEditor can be placed. In this case, there are several options that can be enabled to overcome this limitation:

- Hide ToolBar using header context menu
  Context menu items in header area represent each toolbar available in WebTextEditor. The toolbar can be easily hidden by un-toggling the related menu item. When all toolbars are hidden, the header area will be compressed to 5px height, thus providing a larger editing area for user. The toolbar can easily be displayed afterwards, by toggling the related menu item.

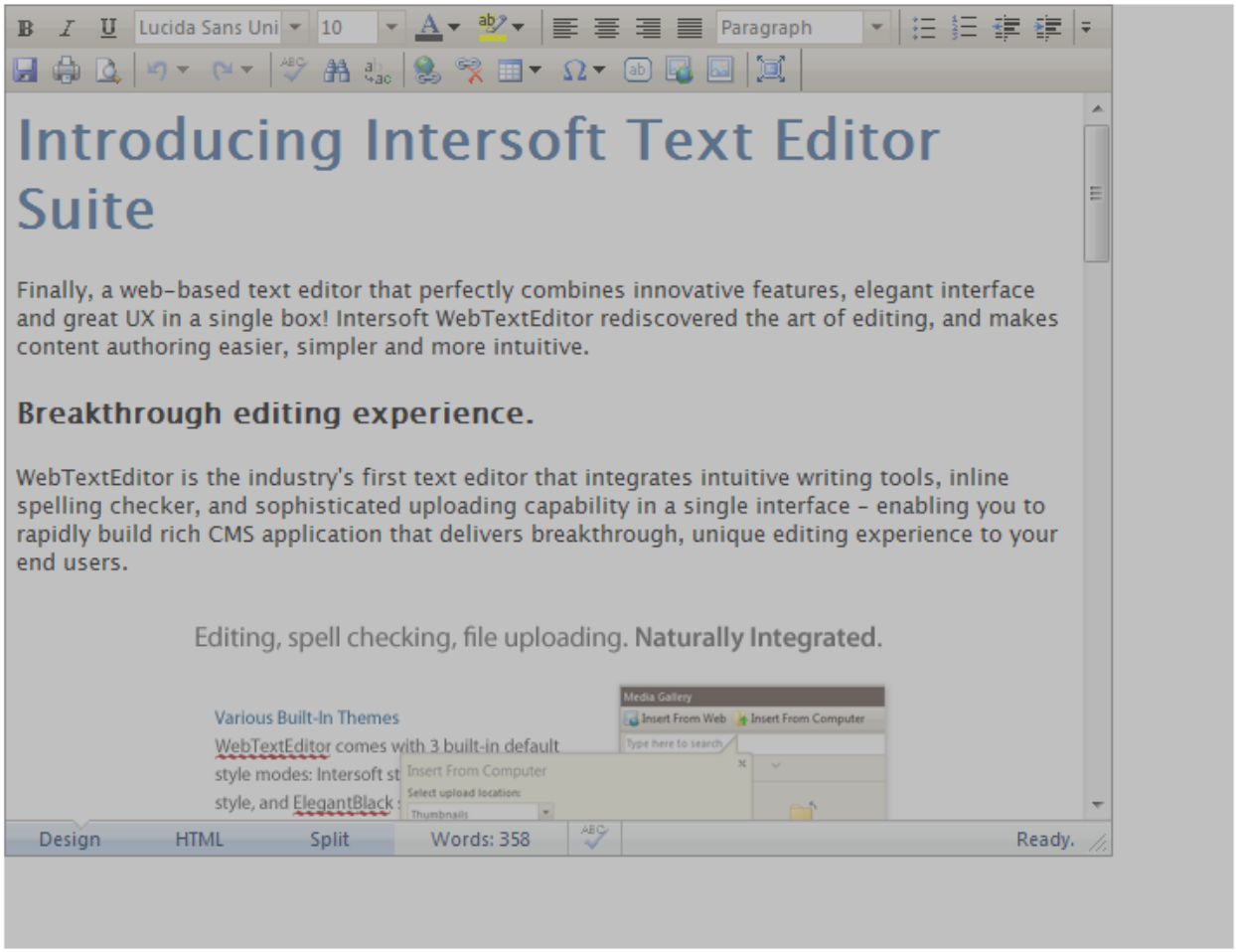


- Allow Resize

  WebTextEditor can be resized to a larger or smaller size by using resize element in main footer area. To enable this feature, simply set *AllowResize* property to one of the following value:

  - Yes: Enable WebTextEditor to be resized horizontally and vertically
  - VerticalOnly: Enable WebTextEditor to be resized vertically
  - HorizontalOnly: Enable WebTextEditor to be resized horizontally
  - No (default value): Disable resize functionality in WebTextEditor

  As shown below, a gray area will be displayed above WebTextEditor when it's being resized. The gray area represents the final size of WebTextEditor after resize action is performed.

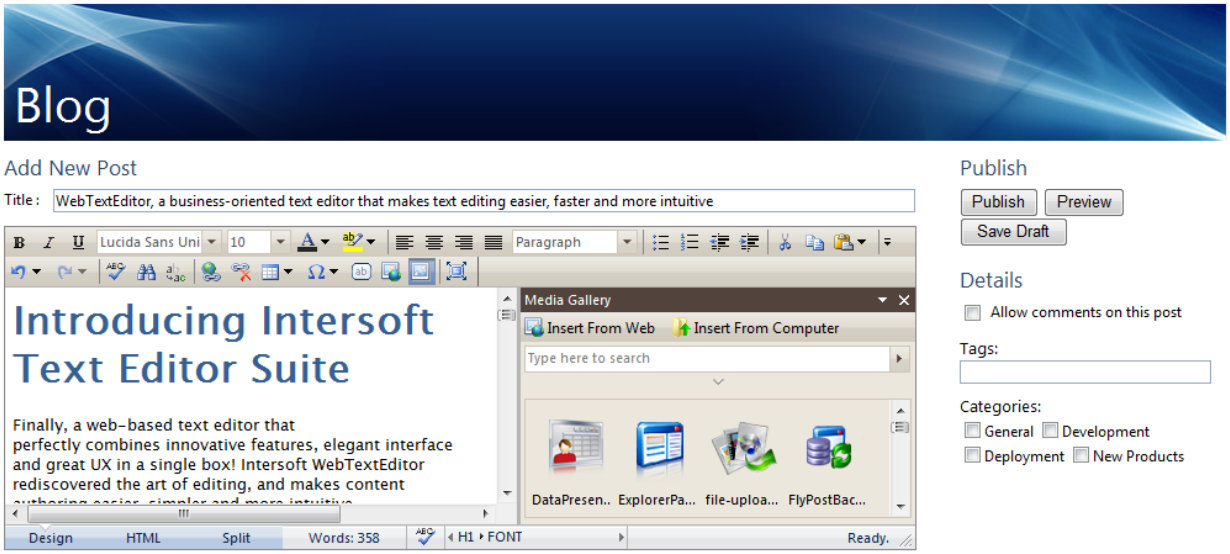


  Related to AllowResize feature, maximum and minimum size can be set to WebTextEditor, using *MaximumSize* and *MinimumSize* properties. When both properties are set, WebTextEditor can only be enlarged until the value of maximum size and reduced to the value of minimum size.

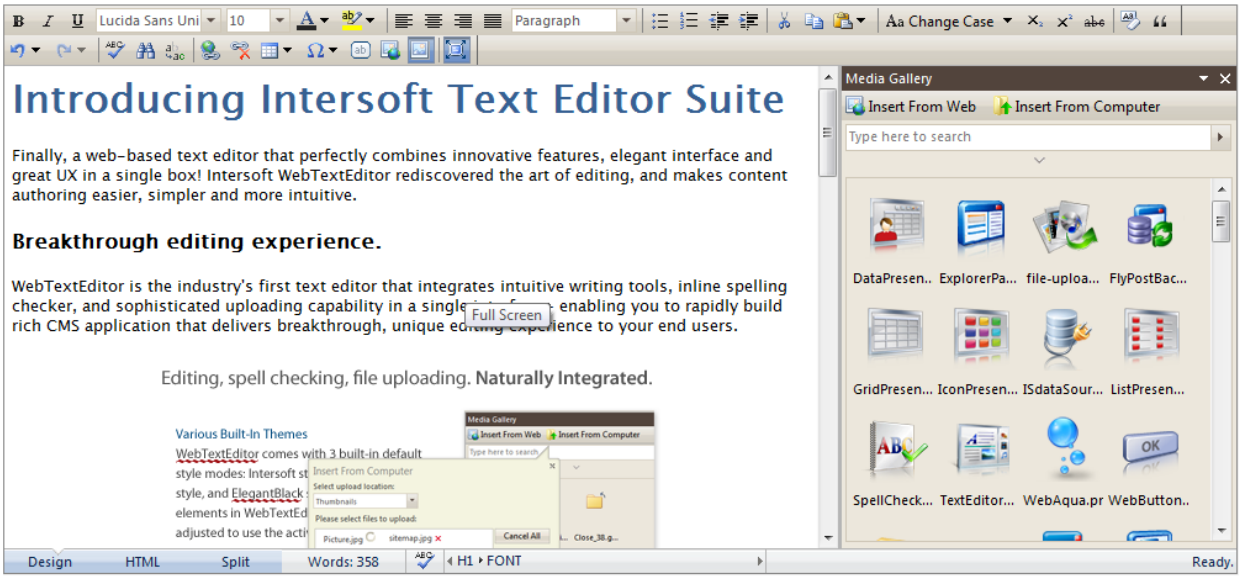
  For example: when MaximumSize property is set to 1024, 768, WebTextEditor can only be enlarged until 1024px width and 678px height. When MinimumSize is set to 500, 400;

WebTextEditor can only be reduced until 500px width and 400px height. When both properties are set to -1, -1, there is no size limitation when WebTextEditor is resized.

- Full Screen
  In certain scenarios where the layout is fixed, resize action might cause inconsistency in the page structured layout. In this case, Full Screen feature provides another option to view WebTextEditor in its maximum size, without modifying the structure of the other elements in the page.



Original WebTextEditor - Full Screen mode is not active



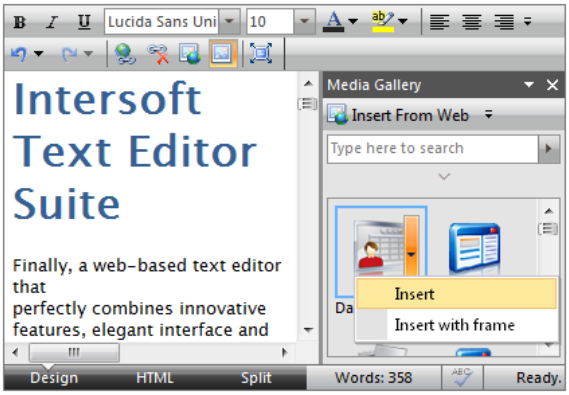
Maximized WebTextEditor - Full Screen mode is active

When Full Screen is activated, WebTextEditor will float above the other elements in the page and its size will be maximized according to the size of the document, providing very maximum editing area for users without changing the layout structure of the page. When user has finished editing, he can deactivate Full Screen mode and WebTextEditor will be placed to its original container using its original size.

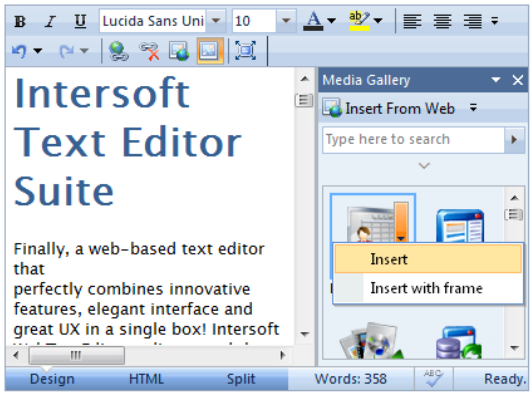
To activate Full Screen mode, simply click FullScreen command in the toolbar. By default, this command is added automatically to toolbar when toolbar mode is not Minimal; however it can be disabled by setting *EnableFullScreen* property to false.

Professional Built-in Themes

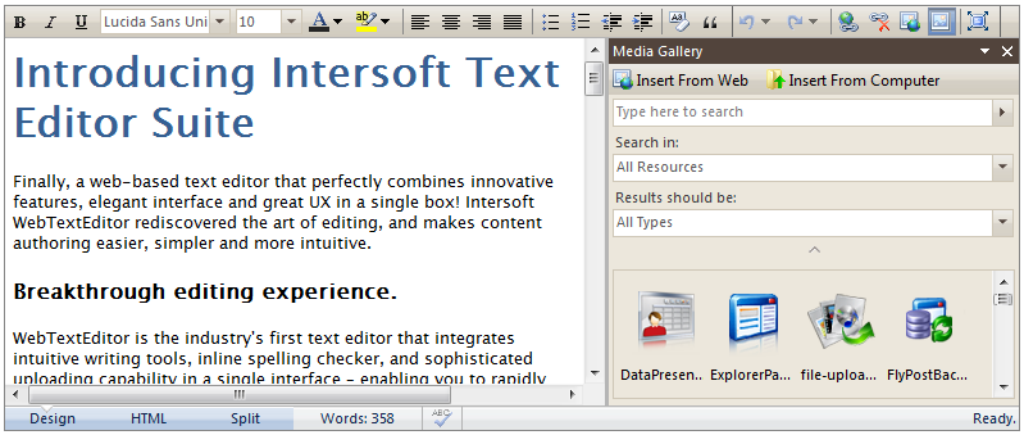
WebTextEditor comes with three built-in themes: Intersoft style, Elegant Blue style, and Elegant Black style. All elements in WebTextEditor besides callout and floating toolbar will be adjusted to use the selected style mode.



Elegant Black Style



Elegant Blue Style



Intersoft Style

By default, Intersoft style will be used as default style mode. To select a different style, simply change *DefaultStyleMode* property.

Allow Style Merging

When using default style mode, all styles in the control will use pre-defined styles according to the selected theme. In previous release, when a style is modified, the default style for that element will be removed and replaced with the custom style. For example: when BackColor=Red is specified in FrameStyle, all the other default style, such as font style, border style, and other styles, specified for FrameStyle will be removed. With this limitation, developer is forced to redefine the removed default style when he wants to modify a style.

Using Allow Style Merging feature available in WebUI Framework, custom style can be added without losing the original default style. So in the above example, the font style, border style and the other styles specified in FrameStyle will be persisted when BackColor=Red is added. Both style definitions will be combined in FrameStyle.

To disable this feature, simply change *AllowStyleMerging* property to False.

Load Custom CSS File

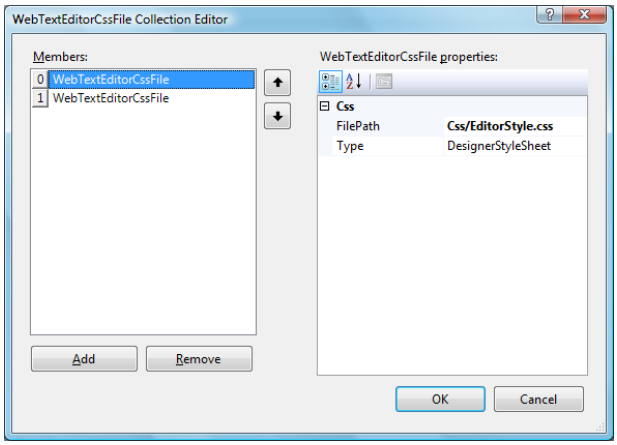
WebTextEditor provides built-in professional styles when working with editor in Design view. For example, tables and image frame will be properly formatted to make the document more readable.

In addition, WebTextEditor also provides flexibility to add custom CSS files to WebTextEditor. The CSS files collection can be added in *RootTextEditor >> CssFiles* property, however when multiple section is enabled, the CSS files collection needs to be added in each section using *[each WebTextEditorSection] >> CssFiles* property.

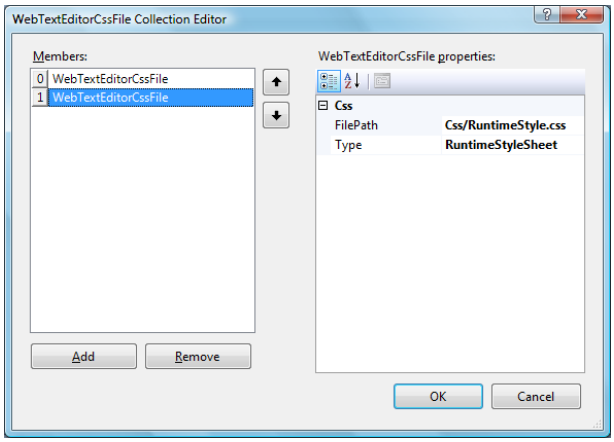
In CSS file object, the following properties needs to be specified:

- FilePath: specifies the path of CSS file
- Type: specifies the type of CSS file. There are two types supported in WebTextEditor: DesignerStyleSheet and RuntimeStyleSheet. Designer-typed CSS files will be loaded when editing the content in Design view, while runtime-typed CSS files will be used when previewing or printing the content.

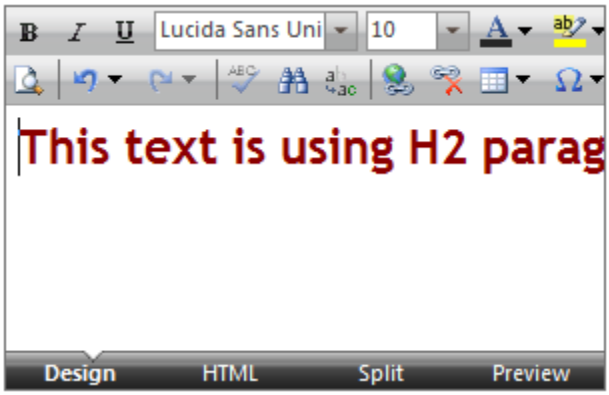
When WebTextEditor is used in blog application or other content management application, usually there are certain format and styles that need to be applied for consistency purpose. Designer-typed styles used in WebTextEditor will not be used when the content is saved, previewed, and printing, making it suitable for this scenario. Instead, the runtime-typed styles will be automatically applied to the content.



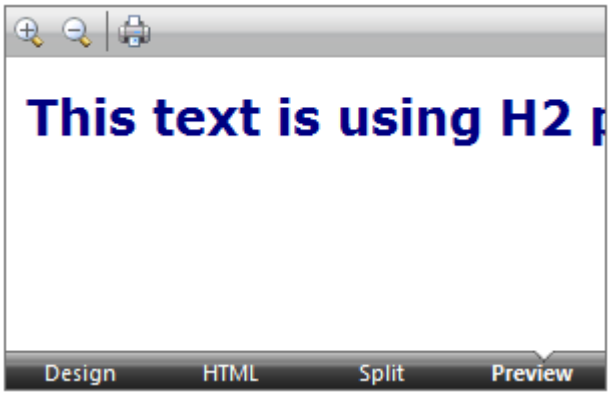
Designer-typed style



Runtime-typed style



Designer-typed style used in Design view



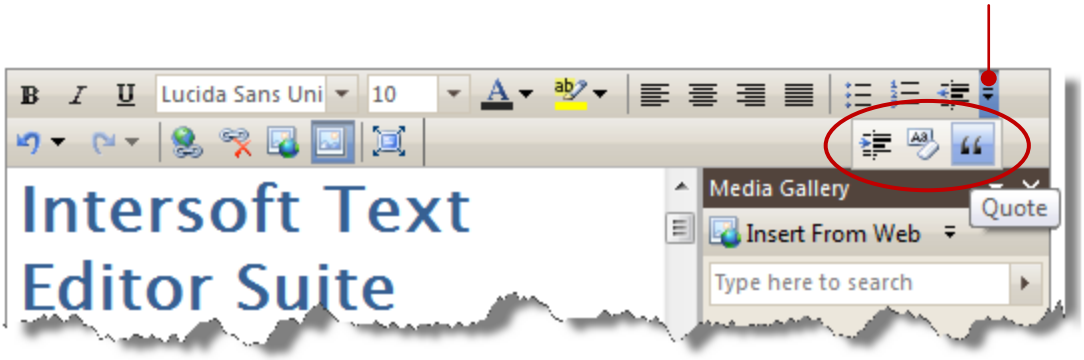
Runtime-typed style used in Preview

ToolBar

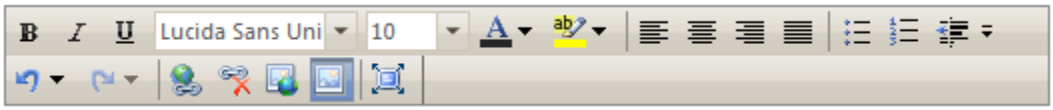
Fluid, Smart ToolBar

One of the unique features of WebTextEditor's toolbar is its ability to detect whether or not the container size is sufficient to display all its commands. When the container size is not sufficient, it can adjust itself to hide the possible commands and placed the hidden commands in MoreMenu container. When the browser is resized and the container size is enlarged, it will detect that more area is now available and show the hidden commands.

Smart toolbar expands and shrinks automatically according to your screen real estate. When the toolbar doesn't have enough space to show all items, the show more button will appear. Click on this button to access more commands.



In WebTextEditor, multiple toolbars can be added to main toolbar area. In Standard toolbar mode, Formatting and Standard toolbars are added to header area of WebTextEditor. With its fluid behavior, toolbar is able to detect whether or not its container can display multiple toolbars in a row. If the container size is not sufficient to display multiple toolbars in a row, it will be shifted to the next row, and so on.

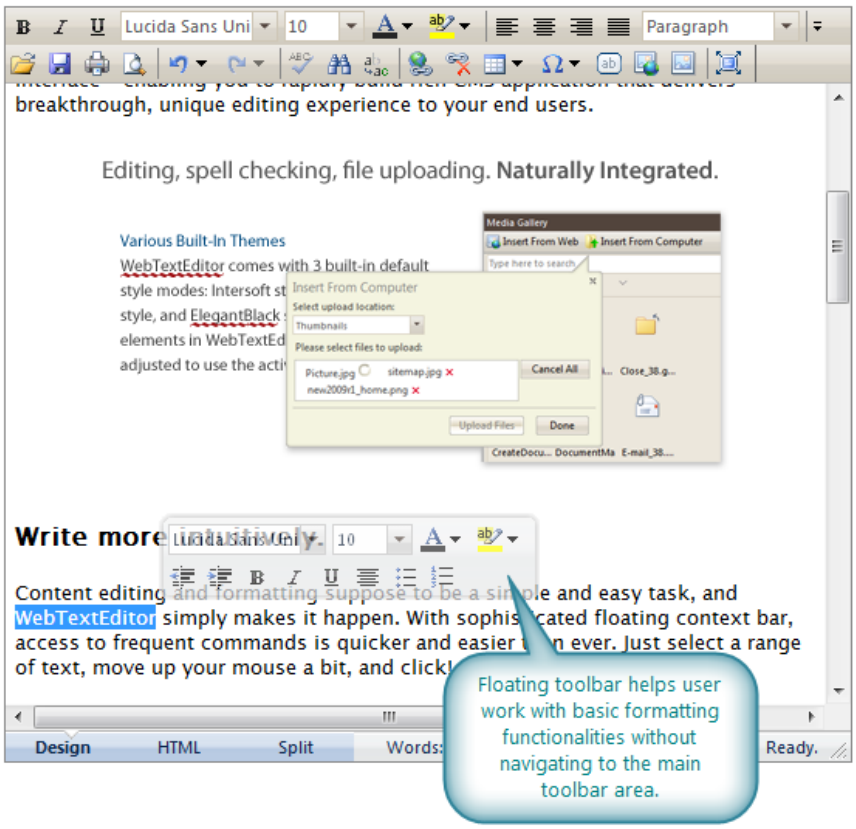


When the container is enlarged to a sufficient size, toolbar will automatically adjust itself and multiple toolbars will be displayed in a row.

Intuitive Word 2007-style Floating ToolBar

To provide a more intuitive editing experience, floating toolbar -- similar to Mini toolbar provided in Microsoft Office Word 2007 -- is natively supported in WebTextEditor. This feature can be enabled by setting *ToolBarSettings >> EnableFloatingToolBar* property to true.

When enabled, user can select text and floating toolbar will be displayed in half opacity near the latest position of cursor. Floating toolbar helps user work with basic formatting functionalities without navigating to the main toolbar area.



Floating toolbar applies independent styles that do not depend on selected theme. It will gradually fade out when user moves further from it and it will be hidden when user hovers on main toolbar area.

Note that floating toolbar will be displayed when a text is selected via mouse. It will not be displayed when user selects a text using keyboard keys.

ToolBar Mode

There are six toolbar modes available in WebTextEditor:

- None. When this mode is selected, toolbar will be disabled. Moreover, when floating toolbar is also disabled, toolbar script will not be loaded.
- Minimal. In this mode, Formatting and Standard toolbars are added. Formatting toolbar will include the following commands: Bold, Italic, Underline, Font Name, Font Color, Align Left, Align Center, Align Right, Bullets, and Numbering. Standard toolbar will include the following commands: Insert from Web and Media Gallery.
- Standard. This mode is the default value for *ToolBarMode* property. In this mode, Formatting toolbar will include the following commands: Bold, Italic, Underline, Font Name, Font Size, Font Color, Text Highlight Color, Align Text Left, Align Text Center, Align Text Right, Align Text Justify,

Bullets, Numbering, Decrease Indent, Increase Indent, Clear Formatting, and Quote. Standard toolbar will include the following commands: Undo, Redo, Hyperlink, Remove Hyperlink, Insert from Web, Media Gallery, and Full Screen.

- Rich. In this mode, Formatting toolbar will include the following commands: Bold, Italic, Underline, Font Name, Font Size, Font Color, Text Highlight Color, Align Text Left, Align Text Center, Align Text Right, Align Text Justify, Bullets, Numbering, Decrease Indent, Increase Indent, Cut, Copy, Paste, Clear Formatting, and Quote. Standard toolbar will include the following commands: Undo, Redo, Spell Checker, Find, Replace, Hyperlink, Remove Hyperlink, Insert Table, Insert Symbol, Insert from Web, Media Gallery, and Full Screen.
- Complete. In this mode, Formatting toolbar will include the following commands: Bold, Italic, Underline, Font Name, Font Size, Font Color, Text Highlight Color, Align Text Left, Align Text Center, Align Text Right, Align Text Justify, Paragraph, Bullets, Numbering, Decrease Indent, Increase Indent, Cut, Copy, Paste, Change Case, Subscript, Superscript, Strikethrough, Clear Formatting, and Quote. Standard toolbar will include the following commands: Undo, Redo, Spell Checker, Find, Replace, Hyperlink, Remove Hyperlink, Insert Table, Insert Symbol, Insert Form Control, Insert from Web, Media Gallery, and Full Screen.
- Custom. In this mode, custom toolbar can be added to WebTextEditor. This topic will be further explained in next section.

ToolBar mode can be changed in *ToolBarSettings >> ToolBarMode* property.

ToolBar Settings

By default, toolbar height is set to 24px and command height is set to 20px. Both properties can be customized using *ToolBarSettings >> Height* and *ToolBarSettings >> CommandHeight* properties.

Several other configurations that can be set in ToolBarSettings:

- *FontNames*: specifies the options that will be listed in Font Names combo. The list should be set as comma separated value.
- *FontSizes*: specifies the options that will be listed in Font Sizes combo. The list should be set as comma separated value.
- *Heading*: specifies the options that will be listed in Paragraph Style combo. The list should be set as comma separated value.
- *LinkOptions*: specifies the options that will be listed in Link combo included in the Hyperlink callout. The list should be set as comma separated value.

Custom ToolBar

Custom toolbar can be added to WebTextEditor using various approaches:

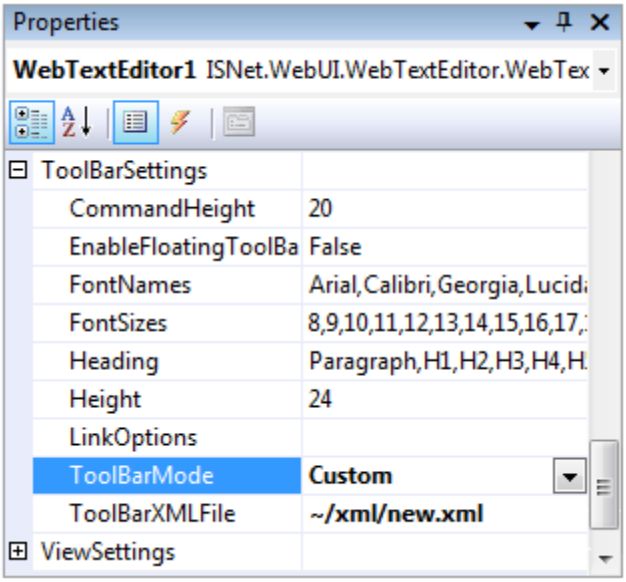
- Using InitializeToolBar server-side event
  In certain scenarios, several commands might need to be added or removed while maintaining the remaining commands that have already been included in the toolbar. InitializeToolBar

server-side event can be used for this purpose. This topic will be further explained in next topic.

- Using XML File

  In Design-time

  To add custom toolbar, simply set *ToolBarSettings >> ToolBarMode* property to Custom, and set the path of custom XML file definition in *ToolBarSettings >> ToolBarXMLFile* property.



  Programmatically

  To add custom toolbar programmatically, use *LoadToolBarsStructureFromXml* method to load the xml file definition.

```
protected void Page_Load(object sender, EventArgs e)
{
    WebTextEditor1.LoadToolBarsStructureFromXml(Server.MapPath("~/xml/new.xml"));
}
```

XML Format for Custom ToolBar

Custom toolbar can be added to WebTextEditor using XML File. The XML file should implement the following format:

- The root node is WebTextEditorToolBarCollection
- To represent each toolbar, create WebTextEditorToolBar and add it to the root node. WebTextEditorToolBar could have Name, Text, and Category attributes. The predefined

Category values are Standard, Formatting, HTML, FloatingTop, FloatingBottom, Preview, and Custom. If you add custom toolbar, you should use Custom category.

- Add ToolCommands node inside each toolbar node
- To represent each command, add WebTextEditorToolCommand to ToolCommands node. The attributes of commands are:
    - Name: specifies the unique name of the command
    - Text: specifies the text that will be displayed in the command
    - DisplayMode: specifies the display mode of the command. Three options are available for this attributes: Text, Image, and TextAndImage.
    - Image: specifies the image path that will be displayed in the command
    - Type: specifies the type of the command. There are various types supported in toolbar: Button, ToggleButton, SplitButton, DropDownButton, and Separator. Note that unique ID must be assigned to Separator-typed command.
    - ToggleGroupName: specifies the name of a toggle group. If multiple commands are included in the same toggle group, the same ToggleGroupName must be assigned to them.
- If the command type is SplitButton or DropDownButton, items can be added to the command. Add Items node inside the command node.
- To represent each item, add WebTextEditorToolItem to Items node. The attributes of an item are:
    - Name: specifies the unique name of the item.
    - Text: specifies the text that will be displayed in the item.
    - Image: specifies the image that will be displayed in the item.

Example:

```
<WebTextEditorToolBarCollection>
  <WebTextEditorToolBar Name="WebTextEditor1_tbCustom" Category="Custom"
Text="Custom 1">
    <ToolCommands>
      <WebTextEditorToolCommand Name="cmdChangeCase" Text="Change Case"
CommandType="ChangeCase" DisplayMode="TextAndImage" Image="
tb_changecase.gif" Type="SplitButton">
        <Items>
          <WebTextEditorToolItem Name="itemUpperCase" Text="UPPER CASE"
Image="tb_uppercase.gif" />
          <WebTextEditorToolItem Name="itemLowerCase" Text="lower case"
Image=" tb_lowercase.gif" />
        </Items>
      </WebTextEditorToolCommand>
      <WebTextEditorToolCommand Name="cmdEmot" Type="ToggleButton"
Text="Emot" DisplayMode="Image" Image="smiley.gif">
        <Items />
      </WebTextEditorToolCommand>
    </ToolCommands>
  </WebTextEditorToolBar>
</WebTextEditorToolBarCollection>
```

Customize toolbar programmatically from server-side

InitializeToolBar server-side event will be invoked when toolbar is being initialized. In this event, commands can be created, added, or removed from toolbar collection. Main toolbar collection, HTML toolbar collection, Preview toolbar collection, and floating toolbar collection will be passed as the arguments of the event.

The following is the code snippet to add a command to toolbar collection.

```
protected void WebTextEditor1_InitializeToolBar(object sender,
ISNet.WebUI.WebTextEditor.WebTextEditorToolBarArgs e)
    {
        WebTextEditorToolBar tb =
e.GetToolBarByCategory(WebTextEditorToolBarCategory.Standard);

        if (tb != null)
        {
            WebTextEditorToolCommandCollection commands = tb.ToolCommands;

            WebTextEditorToolCommand cmd = new WebTextEditorToolCommand();
            cmd.Name = "cmdEmoticon";
            cmd.Text = "Emoticon";
            cmd.ToggleGroupName = "TaskPane";
            cmd.Type = WebTextEditorToolBarCommandType.ToggleButton;
            cmd.DisplayMode = WebTextEditorToolBarCommandDisplayMode.Image;
            cmd.Image = "./images/other/smiley1.gif";
            commands.Add(cmd);
        }
    }
```

The following is the code snippet to remove a command from toolbar collection.

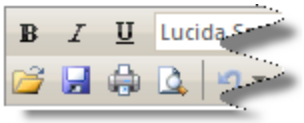
```
protected void WebTextEditor1_InitializeToolBar(object sender,
WebTextEditorToolBarArgs e)
    {
        WebTextEditorToolBar tb =
e.GetToolBarByCategory(WebTextEditorToolBarCategory.Formatting);

        if (tb != null)
        {
            WebTextEditorToolCommandCollection commands = tb.ToolCommands;
            commands.Remove(commands.GetNamedItem("cmdFontColor"));
        }
    }
```

A toolbar collection could contain multiple toolbars. To easily search for a specific toolbar, *GetToolBarByCategory* method in the event's argument can be used. This method can retrieve toolbar object in the collection using the category and name of toolbar.

```
        WebTextEditorToolBar tb =
e.GetToolBarByCategory(WebTextEditorToolBarCategory.Custom, "tbCustom1");
```

Perform Document Properties Actions

As an editing tool, user should be able to import, save, preview, and print a document.



Import a document

When *EnableDocumentImport* property is set to True, Import command will be added to toolbar. When this command is clicked, OnToolBarClick client-side event will be invoked. Various actions can be manually implemented in this event to import a document to editor.

Save a document

When *EnableDocumentSave* property is set to True, Save command will be added to toolbar. When this command is clicked, WebTextEditor will perform FlyPostBack and invoke Save server side event. The editor content will be persisted during this action, so developer can easily grab and save the value and manually.

Print a document

When *EnableDocumentPrint* is set to True, Print command will be added to toolbar. When this command is clicked, user will be prompted to select which printer that will be used to print the content.

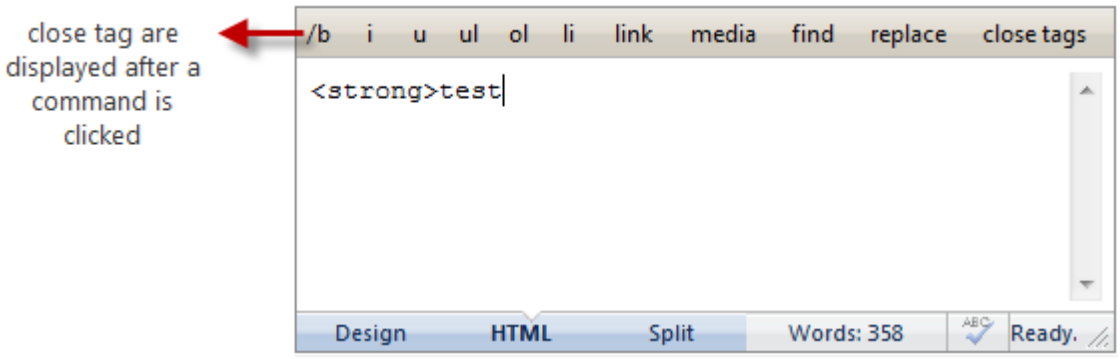
Preview a document

When *ViewSettings >> AllowPreview* is set to True, Preview command will be added to toolbar. When this command is clicked, the view will be switched to Preview mode, where user can preview the content.

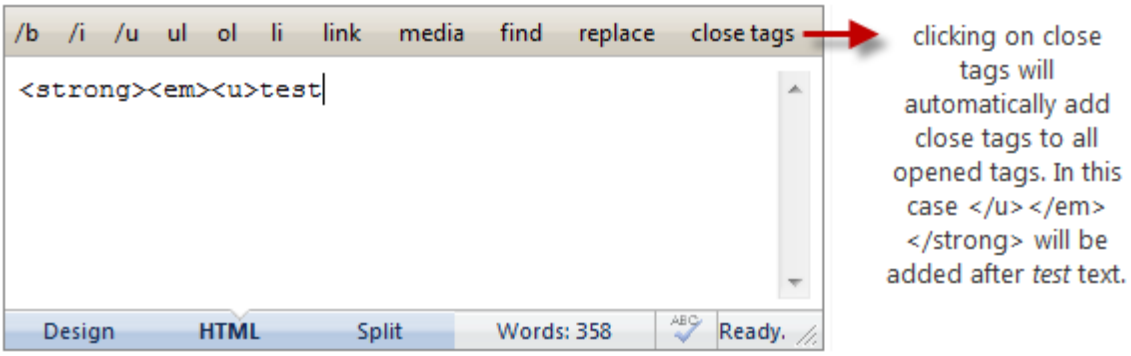
HTML ToolBar

The toolbar commands included in Design view offers editing functionalities that are useful when previewing the content. However those commands do not share the same functionality in HTML view.

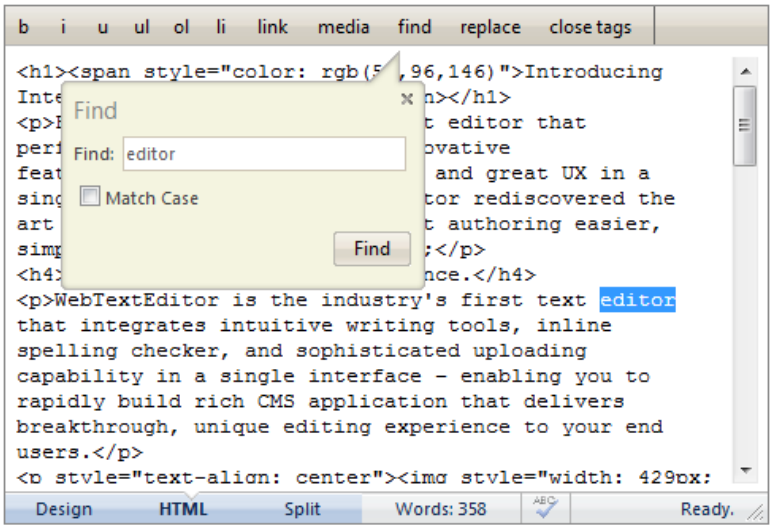
WebTextEditor implements html-related toolbar when viewing in HTML view. When clicked, the command will add the related html tag to editor. After a command is clicked, its text will change to its close tag.



User can easily add some tags and click *close tags* command to add close tags for all previous opened tags.



There is no formatting applied to the content in HTML view, thus making it difficult to find a certain text. WebTextEditor provides Find and Replace functionality in HTML toolbar for easy text searching.



View-based ToolBar

Main toolbar functionality only applies when viewing content in Design view, so when user view HTML view, the main toolbar will be hidden and HTML toolbar will be displayed and vice versa. ToolBar will also be adjusted when user focuses on editor or text area in Split view.

The state of main toolbar will be persisted when it's hidden. So when main toolbar is compressed to 5px height to provide larger editing area in Design view, the state will be persisted when user switches back from HTML view to Design view. Note that user cannot compress HTML toolbar to 5px height.

View

By default, four views are available in WebTextEditor: Design view, HTML view, Split view, and Preview. In Design view, user can view and edit the content visually, while in HTML view, user can view and edit the content in HTML mode. In Split view, the editing area will be split into two areas: the top area will show Design view while the bottom area will display HTML view. When Preview mode is active, user can view the content before it is further processed.

The four views can be enabled and disabled using the following properties in ViewSettings:

- *EnableDesignView*: specifies whether or not Design view is enabled.
- *EnableHTMLView*: specifies whether or not HTML view is enabled.
- *EnableSplitView*: specifies whether or not Split view is enabled.
- *EnablePreview*: specifies whether or not Preview view is enabled. This property is not enabled by default.

Rich Design View

WebTextEditor features a unique design view that provides rich editing capabilities, enabling a true WYSIWYG content authoring experience. The design view includes sophisticated user interfaces such as smart toolbars, lightweight visual elements, intuitive task pane, context menu, and more – allowing users to perform content editing in a more intuitive and natural way.

Please note that WebTextEditor does not allow EnableDesignView, EnableHTMLView, and EnableSplitView properties to be disabled. In that configuration, design view will still be enabled.

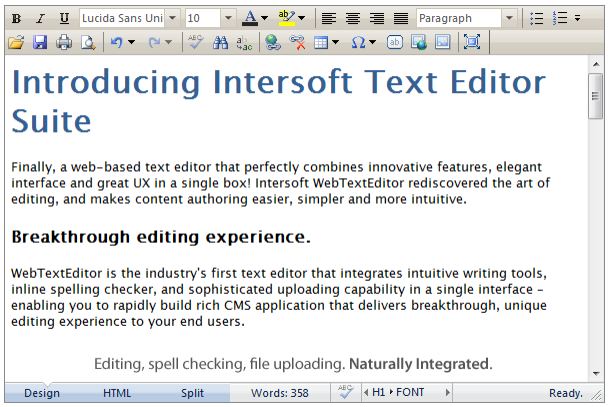
By default, design view will be activated on first load. To activate other view on first load, simply set *ViewSettings >> SelectedViewMode* property to the selected view. A view cannot be disabled if it is set as the selected view mode.

Adaptive HTML View

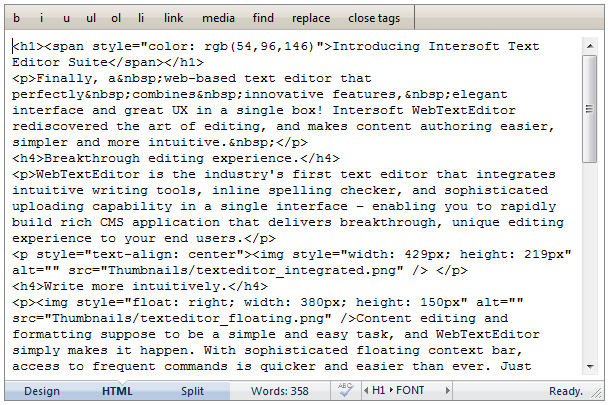
The HTML view allows users to develop more advanced contents by writing html markups directly in the editor. Unique to WebTextEditor, the HTML view displays its own toolbar that contains html-specific and codes writing related commands, minimizing the code-writing efforts.

For instances, click on “bold” command will add tag to the current cursor position, type some text and then click on “close tags” to automatically add the closing tag.

After changes have been made in HTML view, WebTextEditor automatically synchronizes the changes when you switch to Design or other views.



Design view provides rich, WYSIWYG content authoring capabilities

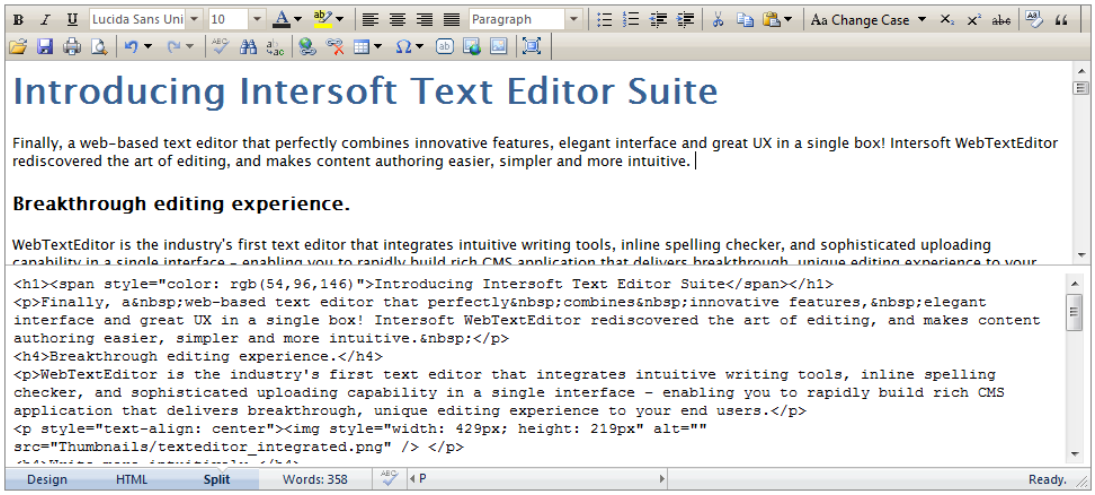


HTML view allows markup editing, suitable for advanced content developer

Intuitive Split View

With Split view, users can work with both visual design and html code in a single interface, eliminating the needs to go back-and-forth between views while improving productivity at the same time. WebTextEditor also automatically synchronizes the changes as user changed the editor’s focus, ensuring each change is properly reflected to other views.

When Split view is enabled, all three views will be enabled even though the Design or HTML view is disabled. This behavior is implemented since Split view requires both views to be enabled.

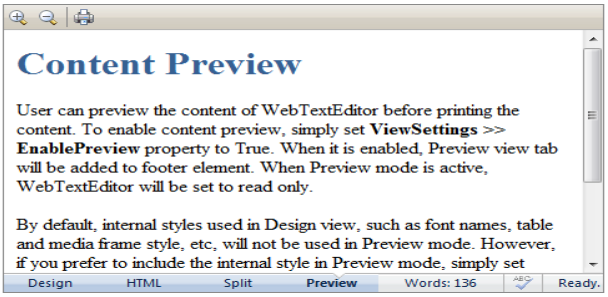


Split view enables users to intuitively work with visual design and html code at the same time.

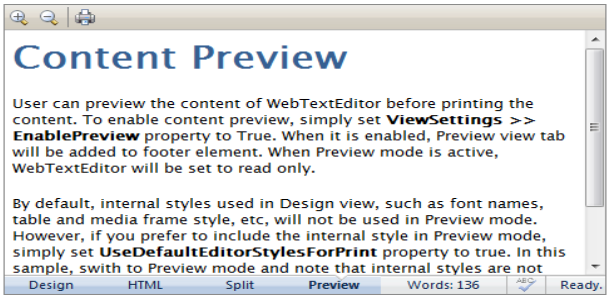
Preview

In Preview mode, user can preview the content before it is printed or further processed. When enabled, Preview tab will be added to the footer row, allowing user to switch to Preview mode easily. Besides using the Preview tab, user can click on Preview command in main toolbar to preview the content.

When editing the content in Design view, several internal styles, such as default font names and size, table and media frame style, etc, are loaded to WebTextEditor, allowing user to differentiate the elements in editor easily. These internal styles will not be loaded automatically when user preview the content in Preview mode. To include the styles when previewing or printing the content, simply set *UseDefaultEditorStylesForPrint* property to true.



Internal styles are not loaded in Preview mode



Internal styles are loaded in Preview mode

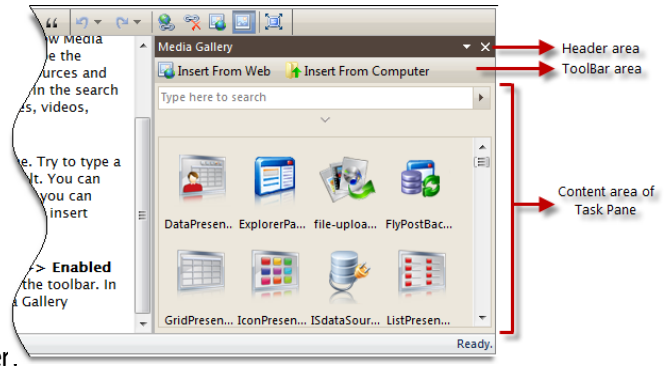
When Preview mode is active, main toolbar will be switched to Preview toolbar, that contains Zoom In, Zoom Out, and Print commands. By default, the zoom value used in WebTextEditor is 10%. This value can be modified in *ViewSettings >> PreviewZoomValue* property.

Task Pane

Innovative Task Pane

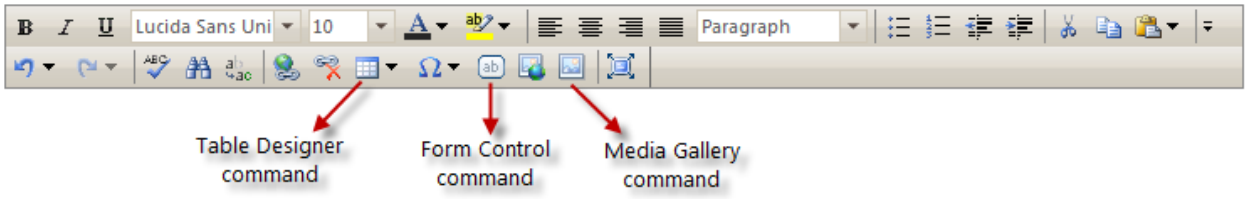
One of the key features in WebTextEditor is its *block-less* user interface concept. With this concept, WebTextEditor allows users to perform various actions without interrupting their editing process.

Task Pane is implemented to support this concept, providing a resizable and extensible container that can be used for multiple purposes without blocking the page. Multiple contents can be included in Task Pane, and users can easily switch

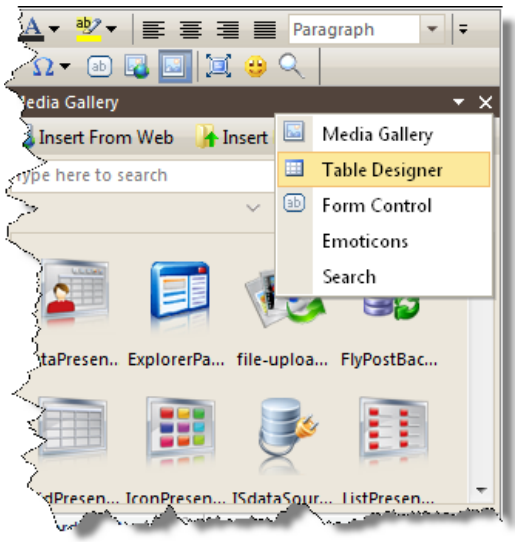


between pane modes and implement individual behavior for each mode.

WebTextEditor includes three built-in task pane modes: Media Gallery, Table Designer, and Form Control. These three panes can be opened from the related commands in main toolbar when *Complete* toolbar mode is activated.

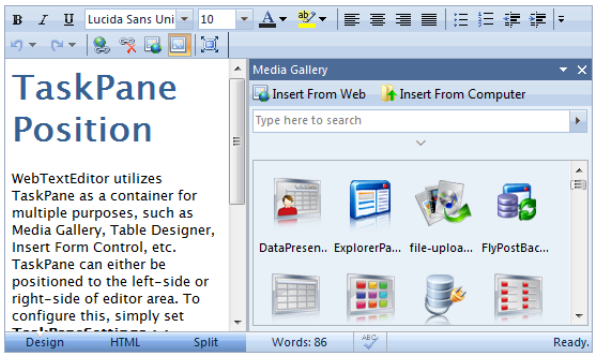


Another alternative is to use context menu options in Task Pane header. All available pane modes will be listed as the context menu items, so user can easily switch between pane modes using this menu.

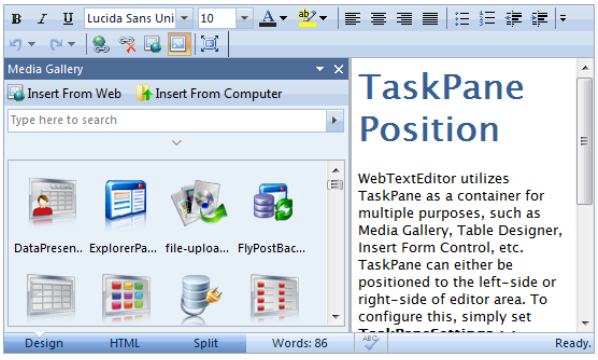


Customizable Task Pane Position

By default, Task Pane is positioned at the right part of WebTextEditor. However, the position can be switched to the left part of WebTextEditor by configuring *TaskPaneSettings* >> *TaskPanePosition* to Left.



Right-positioned Task Pane



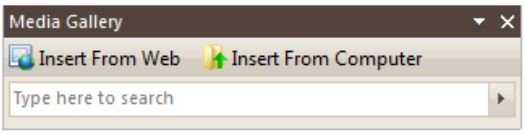
Left-positioned Task Pane

Media Gallery Task Pane

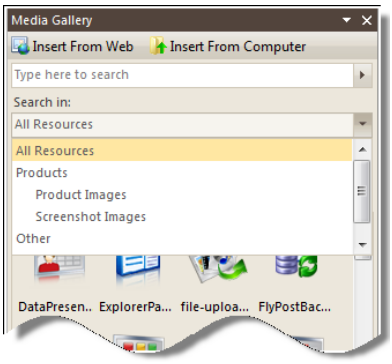
Media Gallery pane is used to search for media and insert them into WebTextEditor. To open Media Gallery pane, user can click on Media Gallery command in main toolbar or choose Media Gallery from Task Pane menu.

Media will be searched based on the search criteria inputted by user:

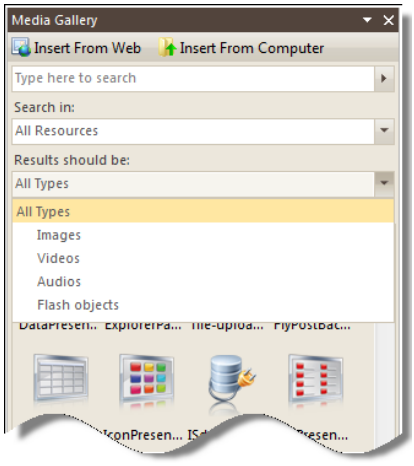
- Search text. When search text is specified, WebTextEditor will search for media that contain the specified text. When search text is not specified, the search will retrieve all media.



- Media Resources. Several resources can be specified in WebTextEditor in hierarchical mode. Media resource refers to an external storage where media are stored. User can determine whether the search action should be performed on all resources or a single resource.

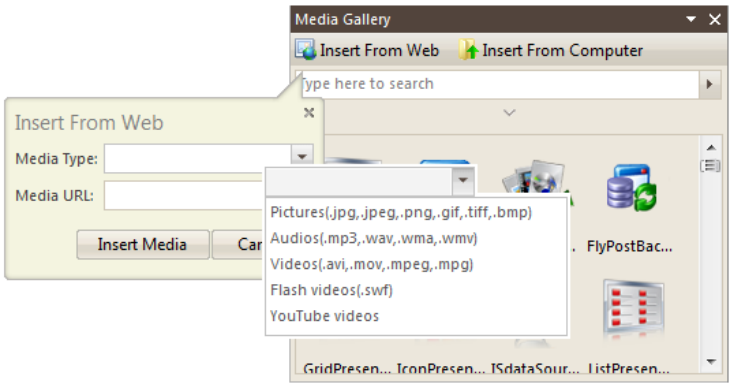


- Result type. The result of the search could vary from images, videos, audios, and Flash objects. User can determine whether the search action should returns all type of results or limited to a certain kind of type.

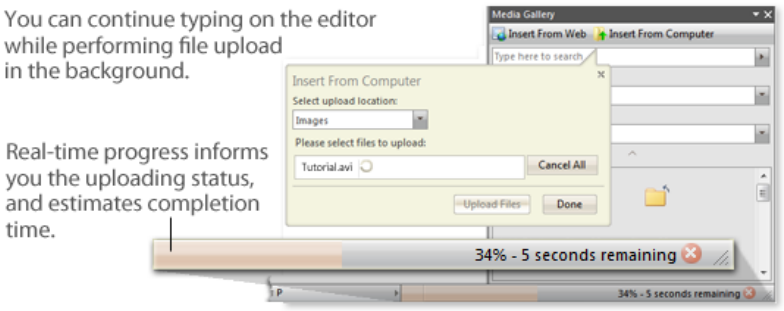


To insert the media to editor, user can simply double click on the media or use the media's context menu. Using media context menu, user can choose whether or not the inserted media be placed in a frame. When frame is used, caption can be added to the image.

If the media is stored in the web, user can use Insert from Web callout to insert the media to editor. Various types of media: pictures, audios, videos, Flash videos, and YouTube videos can be inserted to editor.



In most scenarios, the media is stored in local machine. Users have to upload the file to server before they can make use of it in the editor. WebTextEditor features deep integration with WebFileUploader enabling users to perform files uploading easily and intuitively. Simply click Insert from Computer command to upload the file and insert it in editor.



Media Gallery Settings

In scenarios where Media Gallery is not needed, it can be disabled using *TaskPaneSettings >> MediaGallery >> Enabled* property. When Media Gallery is disabled, the Media Gallery command in main toolbar will also be removed.

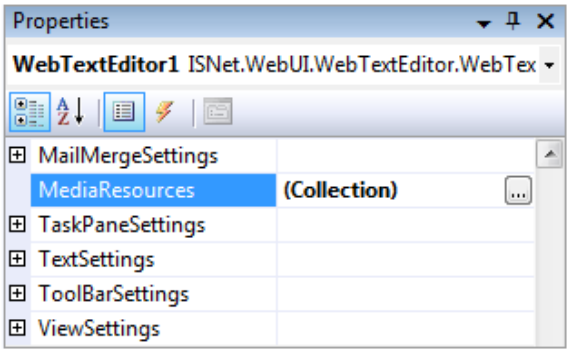
The media retrieved as results of media search action will be displayed as thumbnails in media result container. By default, the thumbnail size is set to 55x55px size. To modify the size of thumbnail, the following properties can be configured: *TaskPaneSettings >> MediaGallery >> ImageHeight* and *TaskPaneSettings >> MediaGallery >> ImageWidth*.

When media search action is performed in multiple media resources, it could result in a large number of media. Retrieving large number of media could affect the performance when rendering and interacting with elements in client-side, thus by design the result is limited to 100 media only. To specify a larger number of search results, simply modify the value in *TaskPaneSettings >> MediaGallery >> MaxMediaRetrieved* property.

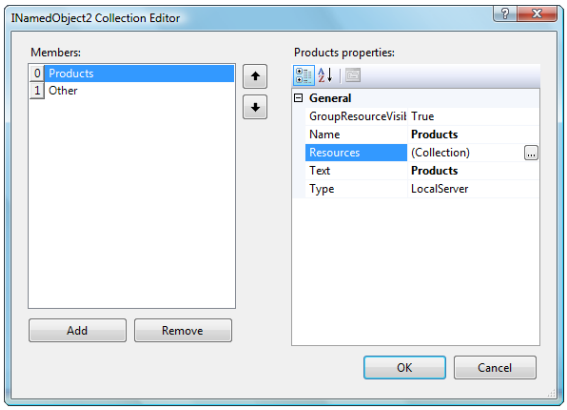
When first opened, media search action will be automatically performed in Media Gallery pane. This behavior can be disabled by setting *TaskPaneSettings >> MediaGallery >> SearchMediaOnLoad* property to False.

Media Resource Concept

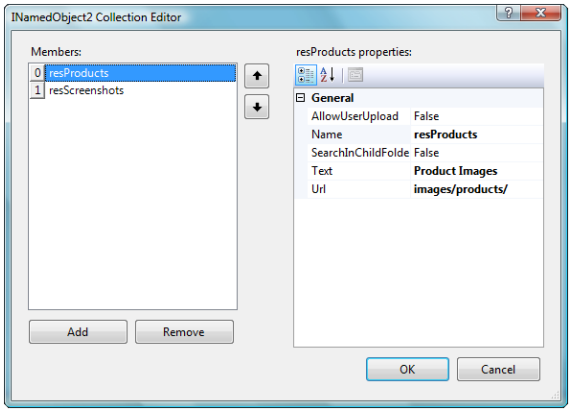
A media resource represents an external storage where media are stored. In this version, only URL storage is supported as media resource. If media are stored in multiple folders, multiple media resources can be created to represents multiple folders. The multiple folders can be also be grouped. User can perform search to individual resource or group.



Media Resource property



Media Resource group



Media Resource

The collection of media resource will be listed as hierarchical layout in Media Resources combo. For example: in the above screenshot, Product Images resource and Screenshot Images resource are grouped in Products group. When user selects Product Images item, the search will be performed in Product Images folder. If Products group is selected, search will be performed against all resources included in Products group, Product Images and Screenshot Images.

Using this approach, search can be performed to a specific resource(s) only, thus filtering the media retrieved as the result.

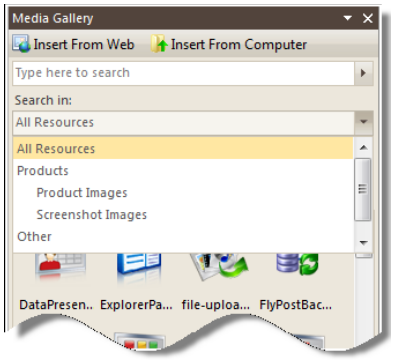
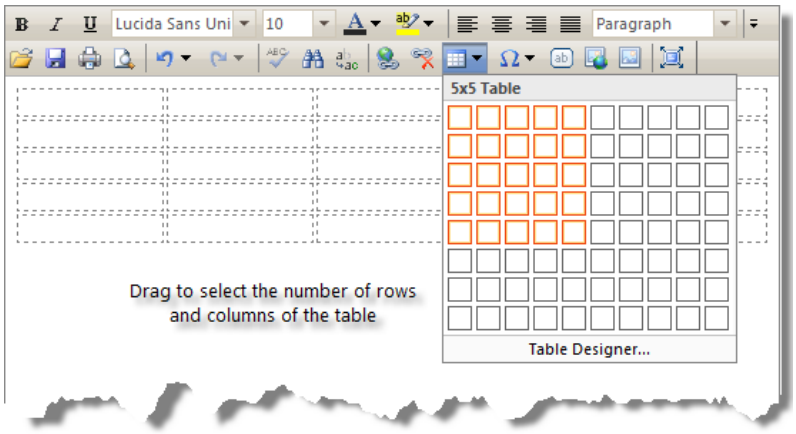


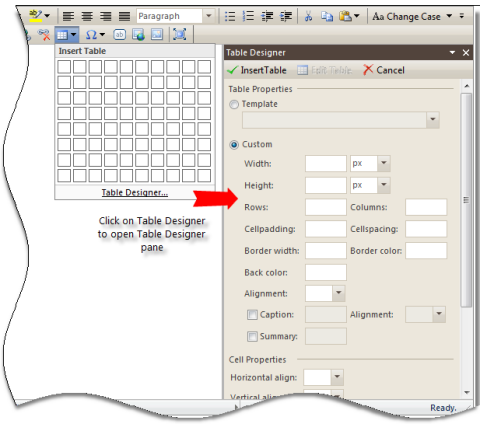
Table Designer Task Pane

In WebTextEditor, user can insert a table using two options:

- Table pop up. Using table pop up, user can drag to select the number of rows and columns of the table. Table will be inserted with 100% width size.

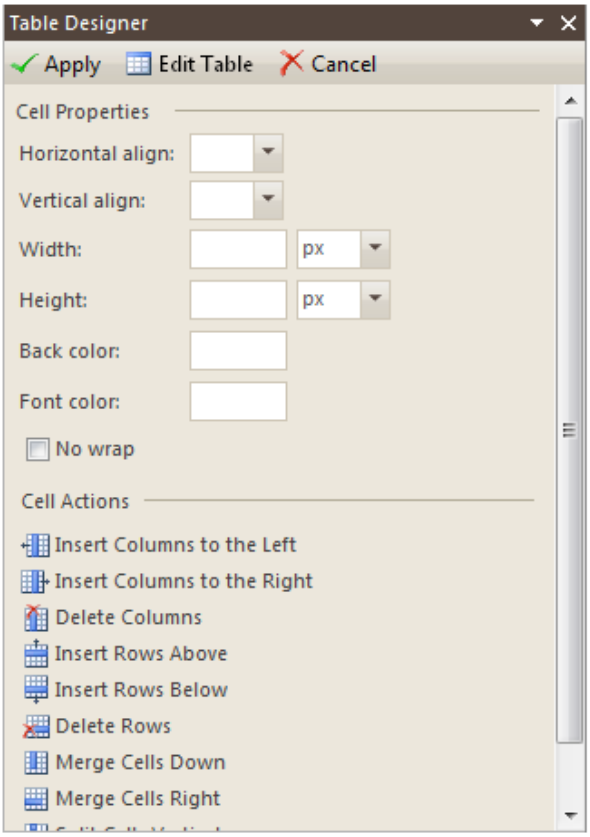


- Table Designer Task Pane
  Using Table Designer Task Pane, user can choose from a list of table layout template available or input the table dimensions and format before inserting the table into editor.



There are three modes in Table Designer pane: Insert Table, Edit Table, and Cell Actions. In Insert Table mode, user can choose a table template from a list available or input table dimensions and formatting. In this mode, user can also specifies cell dimensions and formatting that will be applied to all cells in the table.

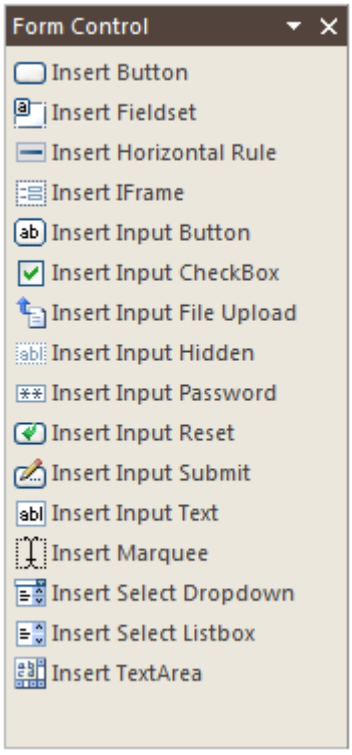
Edit Table mode has the same layout as Insert Table mode, however user is not allowed to modify the number of rows and columns in the table to avoid loss of content if the rows and columns are reduced. User also cannot modify cell properties.



In Cell Actions mode, user can modify cell properties and perform cell actions, such as insert columns to the left, insert columns to the right, delete columns, insert rows above, insert rows below, delete rows, merge cells down, merge cells right, split cells vertical and split cells horizontal. These actions are also available in editor context menu when table is selected.

Not all browsers support table selection, thus navigating to Edit Table mode could be a little bit difficult. In this case, user can utilize *Edit Table* command in pane toolbar to activate Edit Table mode.

Form Control Task Pane



In Form Control pane, user can insert various form controls supported by WebTextEditor. To open Form Control pane, user can click on Form Control command in main toolbar or choose Form Control from Task Pane menu.

Various form controls that can be added to WebTextEditor are: Button, Fieldset, Horizontal Rule, IFrame, Input Button, Input Checkbox, Input File Upload, Input Hidden, Input Password, Input Reset, Input Submit, Input Text, Marquee, Select Dropdown, Select List box, and Text area.

View-Based Task Pane

Unlike Design view, Task Pane is not functional in HTML view, because user cannot view the result visually. So, when user views HTML view, the Task Pane mode will be switched to HTML pane mode, which displays empty content. When user switches back to Design view, the previous state of Task Pane will be persisted. This means, the last opened pane mode, including its values, will be persisted when user switches back to Design view.

This behavior is also implemented in Split view. HTML pane mode will be activated when user focuses on text area section and the last opened pane mode will be activated when user focuses on editor area.

Custom Task Pane

WebTextEditor provides three built-in pane modes: Media Gallery, Table Designer, and Form Control. However in certain scenarios, custom pane mode might need to be implemented. In this case, developers can utilize the extensible pane interface to create their own custom pane mode in Task Pane.

To create a custom pane mode, developer need to create a pane content class that is inherited from TaskPaneContent object and assign a unique ID to the class.

```
function EmoticonPane()
{
	TaskPaneContent.call(this); //inherit from TaskPaneContent object

	this.Id = "Emoticon";
}
```

Several properties can be specified for the custom pane content:

- Type. This is mandatory property that should have unique value that indicates the pane content type implemented.
- Caption: determines the caption of the pane content. The value will be used as the header text in pane content.
- Image: determines the image of pane content. The value will be used as the image of the pane content in Task Pane menu item.
- ShowHeader: determines whether or not the header should be enabled. Default value is True.
- ShowToolBar: determines whether or not the toolbar should be enabled. Default value is True.
- AllowMaximize: determines whether or not the pane is allowed to be maximized. Default value is False.
- AllowClose: determines whether or not the pane is allowed to be closed. Default value is True.
- AllowResize: determines whether or not the pane is allowed to be resized. Default value is True.
- AllowContentOptions: determines whether or not the icon to show available pane content mode should be displayed. Default value is True.

Besides the properties, several interfaces can also be implemented:

- GetContainerElement. This is mandatory function that should be implemented. This function should return the element that will be included in the content area of Task Pane.
- OnCreate: specifies the event that will be invoked when the pane content is being created. The above pane content properties should be specified in this event. If toolbar is enabled, the toolbar commands should be added in this event.
- OnInitialize: specifies the event that will be invoked when the pane content is being initialized. In this event, developer could implement function to initialize default values for input fields in the content area and so on.
- OnClose: specifies the event that will be invoked when the pane content is being closed.
- OnMaximize: specifies the event that will be invoked when the pane content is being maximized.
- OnContentOptionsSelected: specifies the event that will be invoked when an item in Task Pane menu is selected.
- OnResize: specifies the event that will be invoked when Task Pane is being resized.
- OnAfterRender: specifies the event that will be invoked after content rendering process is finished. In this event, developer can register behaviors to elements in the content area or insert new elements to the content area.
- OnShow: specifies the event that will be invoked when the pane content is being shown.
- OnUnload: specifies the event that will be invoked when the pane content is being unloaded. In this event, developer can unregister the behaviors attached to elements in the content area to avoid memory leak. Note that the pane is not unloaded when user switches between pane modes.

After the custom pane object is ready, it should be registered in WebTextEditor using *RegisterPaneContent* function. The custom pane object should be registered when WebTextEditor is initialized, in OnInitialize client-side event.

The custom pane is ready to use. To open the custom pane, developer could add custom command in main toolbar and open Task Pane when the command is clicked.

Show and Hide Task Pane Programmatically

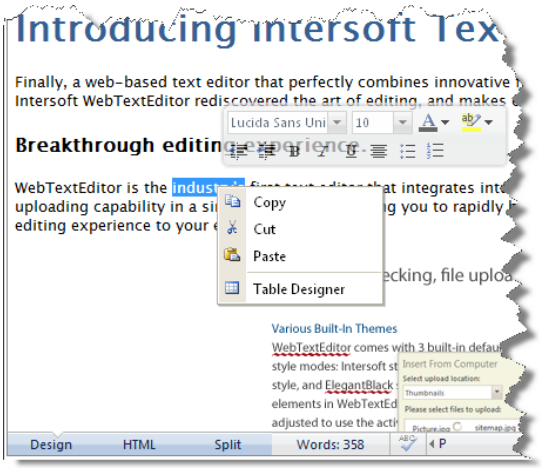
In first load, Task Pane is not opened automatically. User has to click on the related commands in main toolbar to open it. Another alternative is to show Task Pane programmatically using *ShowTaskPane* function. The pane content mode that will be opened should be passed to the function as a parameter.

To hide Task Pane programmatically, *HideTaskPane* function can be used.

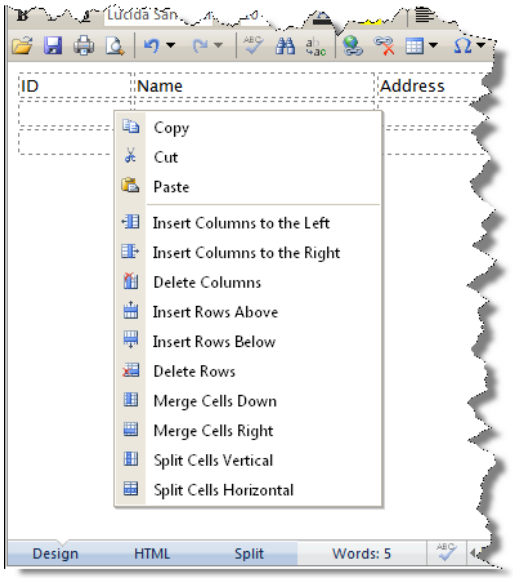
Context Menu

Enable Context Menu

To edit and format the content, user can use toolbar, shortcut keys, and context menu. Context menu in WebTextEditor is not enabled by default. To enable context menu, simply set *EnableContextMenu* property to True.



When enabled, context menu will be displayed when user performs right click in editor area. When user performs right click in table-cell element, cell actions will also be listed in context menu.



When Spell Checker mode is active, the word suggestions will also be listed in context menu, however the context menu used in this mode is not the same menu used in WebTextEditor. The context menu in WebTextEditor will be disabled temporarily when Spell Checker mode is active.

Customize Context Menu

Context Menu can be customized to include custom items or remove existing item. To customize context menu, items can be added or removed in OnEditorContextMenu client-side event.

```
function WebTextEditor1_OnEditorContextMenu(controlId, menuObj)
        {
            var items = menuObj.RootMenu.Items;

            if (items.GetNamedItem("mLookUp") == null)
            {
                items.Add(new WebMenuSeparatorItem());
                items.Add(new WebMenuItem("mLookUp", "Look Up...", null,
OnClick));
            }
        }
```

[HOW-TO: Add Item in Context Menu]

Integration with Intersoft Product

Integration with WebSpellChecker

Integration with WebSpellChecker for spell checking can be done seamlessly and elegantly. To integrate WebTextEditor to WebSpellChecker, simply perform the following steps:

- Add WebTextEditor and WebSpellChecker instances to the page
- In WebSpellChecker instance, set the following properties:

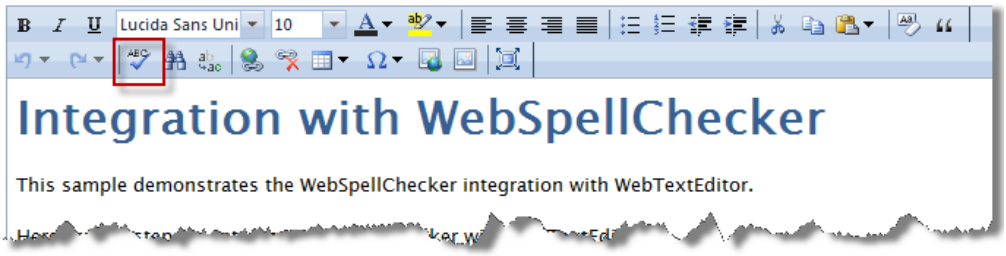
Property	Value
TargetControlId	The ID of WebTextEditor
IntegratedToWebTextEditor	True

When integrated to editor, WebSpellChecker will automatically enable Microsoft Word-style red wave highlight feature. For more information about WebSpellChecker user interface, please refer to WebSpellChecker documentation.

Sophisticated Navigation Experience

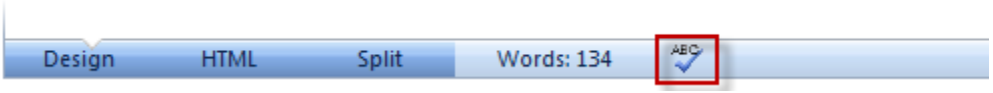
WebSpellChecker introduces sophisticated navigation experience when integrated with WebTextEditor. It also automatically connects to WebTextEditor user interface providing more options for spell checking related commands. You can perform spell checking in several ways such as discussed below.

- Spell checker command in toolbar.



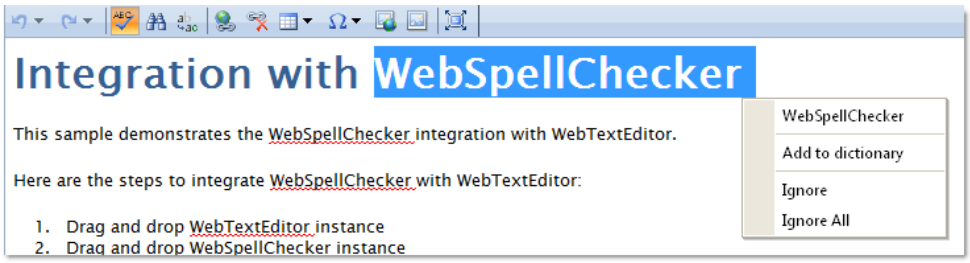
When WebSpellChecker integration is configured properly, the spell checker command in WebTextEditor's toolbar will be enabled. Click on this command to start spell checking which activates spell check editing mode. Click on the command again to exit from spell checking mode.

- Auto-navigate spell checker command in status bar.



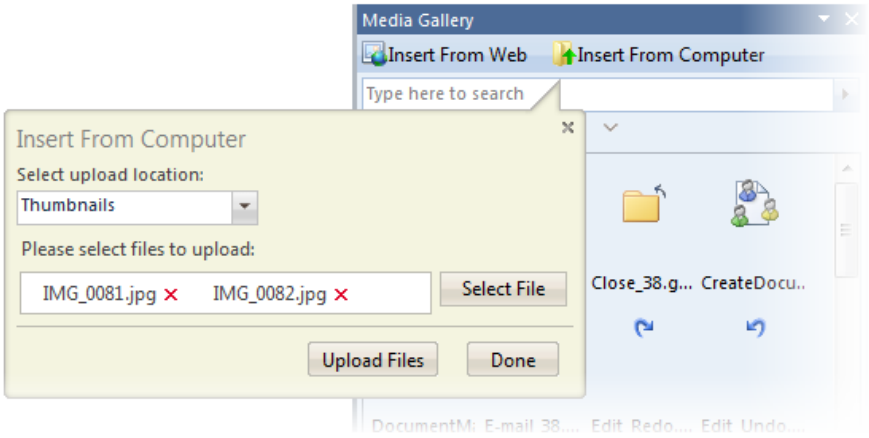
This easy-to-access button provides user with convenient way to perform spell checking. It's particularly useful when the editor is loading in minimal toolbar configuration where spell checker command may not be available in the toolbar.

When clicked for the first time, WebSpellChecker executes spell checking process and automatically focus on the first misspelled word with suggested word list displayed in intuitive context menu interface. Click on the button again to easily navigate to the next misspelled word.

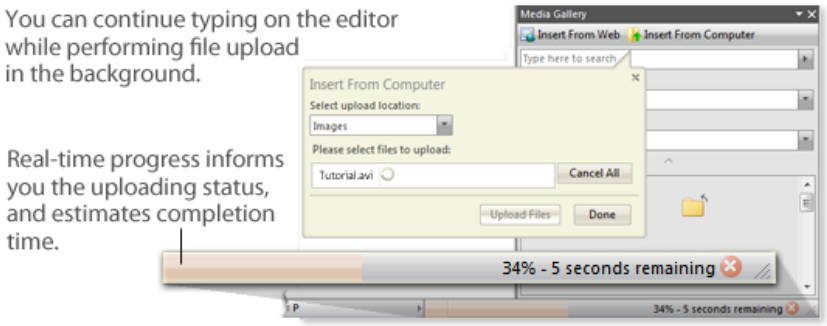


Integration with WebFileUploader

When WebTextEditor is integrated to WebFileUploader, user is able to upload files using *Insert from Computer* command in task pane toolbar. The uploader callout will be displayed when the command is clicked, allowing user to select multiple files to upload.



When *Upload Files* button is clicked, the files will be uploaded. Pink-colored progress bar will be displayed in status bar, indicating the progress status of uploading process. During this process, user can continue editing the content, without having to wait for the uploading process to finish.



When integrated to WebFileUploader, the following configurations can be set in FileUploaderSettings property:

- Enabled: specifies whether or not file uploader is enabled.
- DefaultUploadPath: specifies the default upload path in case none is specified in media resource.
- DefaultUploadMediaResource: specifies the default media resource that is used as default upload path.
- AllowCancel: specifies whether or not the uploading process can be cancelled.
- FilesCount: specifies the maximum number of files that can be uploaded in a single uploading process.
- FileTypes: specifies the types of files that can be uploaded.
- UploadSizeLimit: specifies the maximum size of an uploaded file.
- TotalUploadSizeLimit: specifies the maximum size of total uploaded files.

In addition to the above configurations, the media resources where the files upload is allowed should have **AllowUserUpload** property set to true.

Data Binding

WebTextEditor can be used as an item template of data bound controls. When it is used as the item template, the data can be bound using the same approach used when using .NET controls as item template. WebTextEditor can also be integrated as insert and edit template of data bound controls. Simply bind the data item to **Content** property, as shown in the following snippet.

```
<ISWebTextEditor:WebTextEditor ID="WebTextEditor1" runat="server"
Height="200px" Width="500px" Content='<%# Bind("Notes") %>'>
```

Note that this approach is not yet supported when multiple sections is enabled.

Mail Merge

Mail merge is used when you want to send a set of documents that has the same kind of information but personalized for each recipient. To enable Mail Merge in WebTextEditor, simply set *MailMergeSettings >> EnableMailMerge* to true.

Labels

A collection of labels can be specified in *MailMergeSettings >> Labels* property. When the content is sent, the labels will be replaced with its related value in the data context. The data context can be retrieved from data bound collection or custom collection.

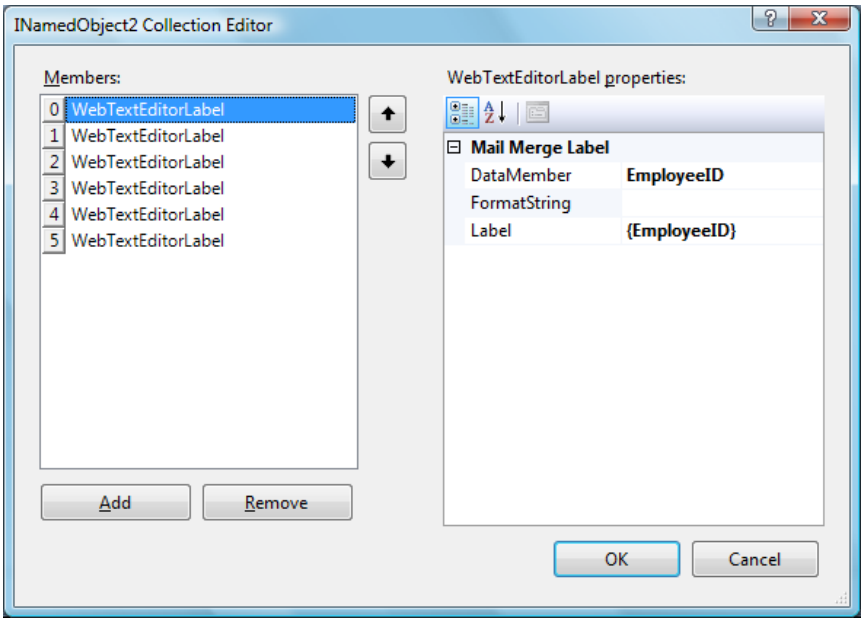
Data Bound Collection

If data bound collection is used, each label will represent a specific field in the data source. Several properties that should be specified are:

- DataSourceID: specifies the id of data source
- DataMember: specifies the data member of data source that will be used to retrieve data

Each label can have the following properties:

- Label: specifies the value of the label
- DataMember: specifies the field in the data context which represents the label
- FormatString: specifies the format string of the label



Each label is related to a specific field in data context.

Custom Collection

If the data context is retrieved from a custom collection, each label will represent a specific property in the custom collection. The custom collection should be specified as the value of Recipients property.

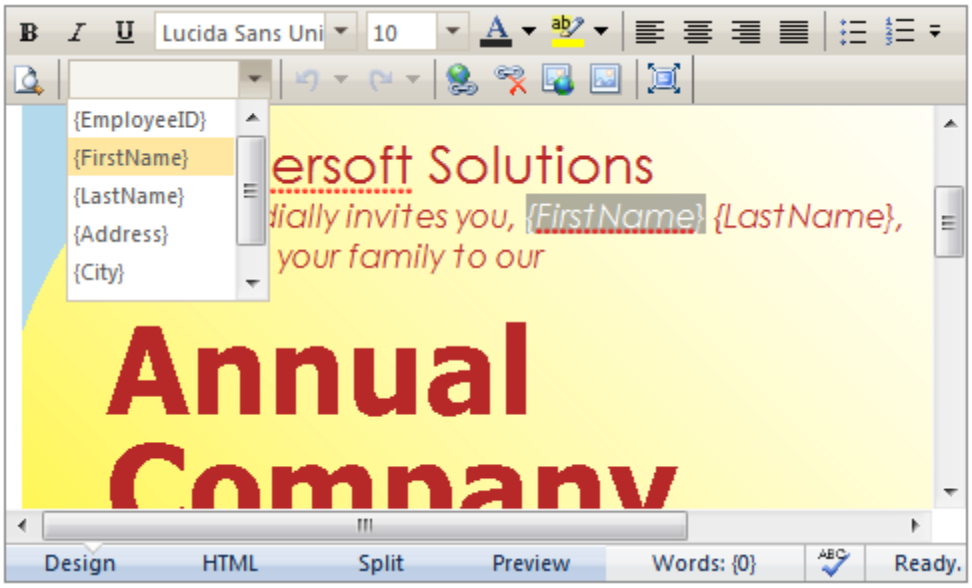
```
WebTextEditor1.MailMergeSettings.Recipients = employees;
```

Each label can have the following properties:

- Label: specifies the value of the label
- DataMember: specifies the property in the data context which represents the label
- FormatString: specifies the format string of the label

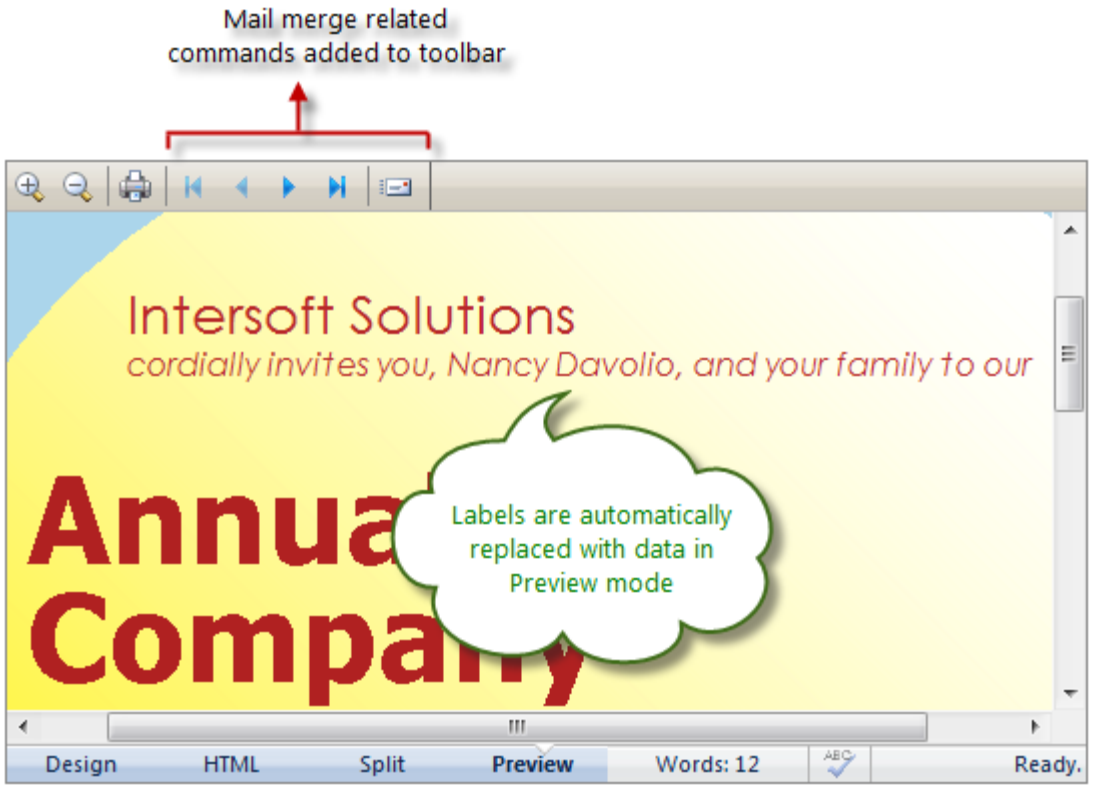
Whether data bound collection or custom collection is used to retrieve the data context, the label replacement will be performed automatically based on the configuration.

The collection of labels in this collection will be added to Label combo in main toolbar. User can easily choose one of the labels and add it to WebTextEditor. As a label indicator in editor, any action that could modify it manually will be prevented.



Preview Mail Merge

When user switches to Preview mode, the labels will automatically be replaced with its related data member in the data context.



Several commands related to mail merge will be added in Preview toolbar. The commands are Send Mail and navigator commands: First, Previous, Next, and Last. User can navigate through the data context using the navigator commands.

Send Mail

When the content is ready, it will be mailed when user clicks on *SendMail* command. Several configurations in *MailMergeSettings* should be configured:

- EmailField: specifies the field in data context that represent the emails of recipients.
- AutomaticSendMail: specifies whether or not the mail will be sent automatically by WebTextEditor using built-in mechanism. Default value is false.
- From: specifies the sender email.
- SubjectExpression: specifies the subject expression of the email. If this property contains label text, it will automatically be replaced with its related field when the mail is sent.
- SMTP: specifies the SMTP server.
- IsHTML: specifies whether or not the content will be sent as HTML-formatted email.
- MailPriority: specifies the priority of the email.

When the emails failed to be sent to some recipients, it will be indicated in the status bar.

The failed email details can be retrieved in *OnSendMailCompleted* client-side event. In this event, three parameters will be passed:

- controlId: specifies the id of WebTextEditor
- failed:specifies the number of emails failed to be sent
- failedEmails: specifies the emails failed to be sent, as a comma-separated value.

Mail Merge-related Server-side Event

Even though labels will be replaced with its related field in the data context, several other processes might need to be performed during mail merge process. To accommodate this need, two server-side events are added to WebTextEditor:

- OnMailMerge. This event will be invoked when mail merge process is performed in each record of data context.

- OnMailMergeCompleted. This event will be invoked when mail merge process has been completed.

When AutomaticSendMail is set disabled, the custom process can be implemented in the following server-side event. Other custom process during send mail process can also be included in these events.

- OnSendMail. This event will be invoked when each mail is about to be sent.
- OnSendMailCompleted. This event will be invoked after all mail has been sent.