Windows Forms Html Editor

The purpose of the Html Editor is to provide Html Editing capabilities within a WinForms control. The control should emulate the operations that are available within a Rich Text control, but have information persisted and restored from an Html BODY element.

Solution Summary

The purpose of the Html Editor is to provide a control that allows for Html editing satisfying the requirements of input for rich text layouts and simple portal type information. Examples of the former are case where the Rich Text control would normally be utilized; documentation, complex descriptions where text formatting is required, correspondences, bulletins, etc. Examples of the latter case are such items as dashboards; news clips, announcements, company references, etc. These are defined by cases where complex layouts are required that may include images and links.

Goals

High level design goals are:

1. Provides robust WYSIWYG editing capabilities whose contents are persisted in HTML format.
2. Is easily reusable in other projects.
3. Provides methods for saving HTML files to and loading files from disk (with the appropriate security demands).

The basic operations of the control are thus defined as:

Standard Text Editing

1. Support basic formatting commands such as Bold, Italic, Underline, Strikeout, Font Name, Font Size, Font Color, Justification (Left, Right, and Center), Bullets and Number Lists.
   Dialogs should be presented to the user for modifying Font and Color attributes.
2. Provide for standard Cut, Copy, Paste, Undo, Redo, Select All, and commands.
3. Allow for the inserting and removing indentation.
4. Allow the inclusion of images along with alternative text and text alignment options.
5. Allow for the insertion of web links (Anchor tags), including the definition of the target frame.
6. Allow for the insertion of a horizontal line for text separation.
7. Provide a Find and Replace mechanisms. This dialog should highlight the appropriate text upon a find, and support a Replace All operation.
8. Provide an integrated toolbar to perform the standard text editing functions, and other essential functions (as listed in the above points).
9. Allow for the Insert mode (overwrite), word wrapping options to be toggled, and the visibility of scroll bars to be defined.
10. Allow the use of context menus that include all the required text formatting commands. The context menu should be sensitive to the user’s selection.
11. Allow for the insert, removal, and property definition of tables.

Body Properties

12. Have the ability to simply set the text of the document body, the inner text of an Html Document; a browsable designer property.
13. Allow for the assignment of the complete Body element (Body outer Html), preserving and body properties. Also allow for the assignment of the Body contents, Body inner Html.

14. Support the inclusion of Headings and Formatted blocks of text. This operation should be able to be performed in reverse; set to Normal style.
15. Have the ability to define the default body background and foreground colors.
16. Allow for the ability of the Html content to be viewed or edited directly.
17. Allow for the pasting of Text and Html Markup.

External Behavior

18. Allow a reference to a stylesheet to be applied to the document at runtime. The purpose is to allow the definition of a corporate wide stylesheet that all documents should reference for standardizing fonts, colors, etc.
19. Allow a reference to a script source file to be applied to the document at runtime. The purpose is to allow the use of a corporate script file that can be used for handling links requiring programmatic redirection.
20. Allow for the ability to ensure all links are forwarded to a new browser window; and not rendered within the window containing the original link.
21. Allow a document to be loaded by a given URL.

Non-Goals

The Html Editor is not designed to provide similar functionality to Html Editor Products; such as Microsoft FrontPage. For complex layout requiring Styles, Absolute Positing, Frames, Multi-Media, etc, these products should be utilized.

Operations that the control does not support are thus defined as:

1. The toolbar is non interactive in that it does not toggle Bold, Underline, and Italic state based on the user selection.
2. Support is only included for a single Font selection and not Font Families.
3. The use of Bookmarks is not supported; including the insertion of named element to which bookmarks can be linked.
4. Support for 2D-Position, Absolute Position, and Live Resize is not included.
5. Multiple Selections of items is not supported and all operations are based on a single selected control.
6. Simple Font properties are used rather than style attributes. The inclusion of style attributes brings around complexity regarding the use of Span tags.
7. Currently when one uses the Tab key the text will not be indented but rather the default Tab form behavior is active; the next control is selected.
8. There was the option to have the control be Tab driven; supporting Design, Edit Html, and Preview. This would then have made the control look more like a fully-functional Html editor rather than a replacement to the Rich Text Box.

Html Editor API

Properties

Textual Properties:

Name	Type	Browsable	Description
InnerText	String	DesignOnly	Defines the Initial Body Text from the designer; Ignores Html Markup.
InnerHtml	String	No	The Inner Html of the Html Body content.
BodyHtml	String	No	The Outer Html of the Html Body content. Used to define new Body contents. Once set the ReadOnly and Body Properties should be reset based on there original values.
DocumentHtml	String	No	The complete Html including the HEAD and BODY tags. Property is Read Only.
SelectedText	String	No	Insert the given Text over the selected text; or current insertion point.
SelectedHtml	String	No	Insert the given Html over the selected text; or current insertion point.
DocumentUrl	String	No	The Url that was used to load the current document; if any.

Body Properties:

Name	Type	Browsable	Description
BodyBackColor	Color	Yes	Defines the Background Color of the Html content. Default value should be based on the associated form property; Ambient.
BodyForeColor	Color	Yes	Defines the Foreground Color of the Html content. Default value should be based on the associated form property; Ambient.
BodyFont	HtmlFontProperty	Yes	Defines the base font attributes for the Html content. Default value should be based on the associated form property; Ambient.

Runtime Display Properties:

Name	Type	Browsable	Description
ScrollBars	DisplayScrollBarOption	Yes	Controls the display of the Scroll Bars.
AutoWordWrap	Boolean	Yes	Controls the Auto Wrapping of content.
NavigateAction	NavigateActionOption	Yes	Window into which to load a

Name	Type	Browsable	Description
			Href link.
ReadOnly	Boolean	Yes	Marks the content as ReadOnly; not editable.
ToolbarVisible	Boolean	Yes	Visible state of the Html editor toolbar.
EnableVisualStyles	Boolean	Yes	Indicates if the Control Flat Style is set to System or Standard for all dialogs
ToolbarDock	DockStyle	Yes	Defines the docking location of the editor toolbar.

Operations

Document Processing Operations:

Prototype	Description
void OpenFilePrompt()	Present the user with the system OpenFileDialog from which they can select an Html file. Upon selection load the contents into the Html body.
void SaveFilePrompt()	Present the user with the system SaveFileDialog from which they can select an Html file. Upon selection persist the Html body as an Html file.
void NavigateToUrl(string url)	Allows the loading of a document through Navigation of a Page Href.
void LoadFromUrl(string url)	Allow the loading of a document through obtaining the Html stream and setting the editor body.
void LoadFromFile (string filename)	Allows the loading of a document through loading of a given file name.
void LinkStyleSheet (string stylesheetHref)	Allow the definition of a style sheet Href to be used for the document. Only a single stylesheet is supported to allow for standard visual templates to be applied.
string GetStyleSheetHref()	Return to the user the style sheet Href being used.
void LinkScriptSource (string scriptSource)	Allow the definition of a script element that is to be used by the document. Only a single script source is supported to allow for standard processing templates to be applied.
string GetScriptBlockSource()	Return to the user the script block source being used.
void HtmlContentsEdit()	Allow the user to edit the contents of the Body Html. A dialog is to be presented with the Body Html upon which the appropriate contents are replaced.
void HtmlContentsView()	Allow the user to view the complete Html markup; including the Head and Body tags.
void DocumentPrint()	Print the Html text using the document print command; print preview not being supported.
void ToggleOverWrite()	Toggle the document overwrite mode.

Document Text Operations:

Prototype	Description
void TextCut()	Cut the currently selected text to the clipboard.
void TextCopy()	Copy the currently selected text to the clipboard.
void TextPaste()	Paste the currently selected text from the clipboard.
void TextDelete()	Delete the currently selected text from the document.
void TextSelectAll()	Select the entire document contents.
void TextClearSelect()	Clear the current document selection.
void EditUndo()	Undo former editing operation.
void EditRedo()	Redo former Undo.

Selected Text Formatting Operations:

Prototype	Description
void FormatFontName(string name)	Set the Font name of the selected text.
void FormatFontSize (HtmlFontSize size)	Set the Font size of the selected text.
void FormatBold()	Set the Font Bold attribute of the selected text.
void FormatUnderline()	Set the Font Underline attribute of the selected text.
void FormatItalic()	Set the Font Italic attribute of the selected text.
void FormatSubscript()	Set the Font Subscript attribute of the selected text.
void FormatSuperscript()	Set the Font Superscript attribute of the selected text.
void FormatStrikeout ()	Set the Font Strike Through attribute of the selected text.
void FormatFontIncrease()	Increase the font size of the selected text font by one Html font size.
void FormatFontDecrease()	Decrease the font size of the selected text font by one Html font size.
void FormatRemove()	Remove any formatting tags from the selected text.
void FormatTabRight()	Tab the current selected line to the right; Indent.
void FormatTabLeft()	Tab the current selected line to the left; Outdent.
void FormatList (HtmlListType listtype)	Mark the selected text as an ordered or unordered list.
void JustifyLeft()	Define the font justification of the selected text as Left justified.
void JustifyCenter()	Define the font justification of the selected text as Center justified.
void JustifyRight()	Define the font justification of the selected text as Right justified.

Object Insert Operations:

Prototype	Description
void InsertLine()	Insert a horizontal Line at the selected insertion point; or over

Prototype	Description
	the selected text.
void InsertImage (string imageLocation)	Insert an Image at the selected insertion point; or over the selected text/control.
void InsertImagePrompt()	Insert an Image at the selected insertion point, or over the selected text/control, prompting the user for the Image attributes. If an Image is currently selected the attributes should be defined based on the selection.
void InsertLink(string href)	Insert a Link Reference at the selected insertion point; using the selected text as the description text.
void InsertLinkPrompt()	Insert a Link Reference at the selected insertion point, or over the selected text, prompting the user for the Href attributes. If a Href is currently selected the attributes should be defined based on the selection.
void RemoveLink()	Remove any link references contained within the selected text.

Text Insert Operations:

Prototype	Description
void InsertHtmlPrompt()	Insert the given Html over the selected text; or current insertion point; prompting the user for the Html.
void InsertTextPrompt()	Insert the given Text over the selected text; or current insertion point; prompting the user for the Text.
string[] GetFormatBlockCommands()	Returns the acceptable values for the possible Format Block commands.
void InsertFormatBlock (string command)	Formats the selected text with the given Format Block; possible values are defined by the GetFormatBlockCommands.
void InsertFormattedBlock()	Formats the selected text with the Formatted Format Block; PRE tag.
pulic void InsertNormalBlock()	Removes and Format Block commands from the selected text.
void InsertHeading (HtmlHeadingType headingType)	Formats the selected text with the Heading Format Block; H1, H2, H3, H4, H5 tag.

System Prompt Dialog Functions:

Prototype	Description
void SystemInsertImage()	Allows the insertion of an Image using the system dialogs.
void SystemInsertLink()	Allows the insertion of a Href using the system dialogs.

Font and Color Processing Operations:

Prototype	Description
void FormatFontAttributes (HtmlFontProperty font)	Using the mshtml commands apply the font attributes to the selected text.
void FormatFontColor(Color color)	Using the mshtml commands apply the color

Prototype	Description
	value to the selected text.
void FormatFontAttributesPrompt()	Present to the user a dialog for font attributes; basing initial values from the currently selected text. Upon completion apply the font attributes to the selected text.
void FormatFontColorPrompt()	Present to the user a dialog for color selection; basing initial values from the currently selected text. Upon completion apply the color value to the selected text.
HtmlFontProperty GetFontAttributes()	Determine the font attributes of the currently selected text.
bool IsFontBold()	Determine if the selected text is Bold.
bool IsFontUnderline()	Determine if the selected text is Underline.
bool IsFontItalic()	Determine if the selected text is Italic.
bool IsFontStrikeout()	Determine if the selected text is Strikeout.
bool IsFontSuperscript()	Determine if the selected text is Superscript.
bool IsFontSubscript()	Determine if the selected text is Subscript.

Find and Replace Operations:

Prototype	Description
void FindReplacePrompt()	Present the user with a dialog to allow standard find and replace operations on the document text.
void FindReplaceReset()	Initializes and Find and Replace operations; to being at the start of the document.
bool FindFirst(string findText)	Finds the first occurrence of the given string within the document. Partial words are acceptable and case is ignored.
bool FindFirst(string findText, bool matchWhole, bool matchCase)	Finds the first occurrence of the given string within the document.
bool FindNext(string findText)	Finds the next occurrence of the given string within the document; based on the last successful find operation. Partial words are acceptable and case is ignored.
bool FindNext(string findText, bool matchWhole, bool matchCase)	Finds the next occurrence of the given string within the document; based on the last successful find operation.
bool FindReplaceOne(string findText, string replaceText);	Replaces the given occurrence of the given string within the document; based on the last successful find or replace operation. Partial words are acceptable and case is ignored.
bool FindReplaceOne(string findText, string replaceText, bool matchWhole, bool matchCase)	Replaces the given occurrence of the given string within the document; based on the last successful find or replace operation.

Prototype	Description
bool FindReplaceAll(string findText, string replaceText)	Replaces all occurrences of the given string within the document; based on the last successful find or replace operation. Partial words are acceptable and case is ignored.
bool FindReplaceAll(string findText, string replaceText, bool matchWhole, bool matchCase)	Replaces all occurrences of the given string within the document; based on the last successful find or replace operation.

Table Processing Operations:

Prototype	Description
void TableInsertPrompt()	Insert a Table at the selected insertion point, or over the selected text/control, prompting the user for the Table attributes. If a Table is currently selected the table attributes should be defined based on the selection.
bool TableModifyPrompt()	Modify the Table at the selected insertion point, or the selected table, prompting the user for the Table attributes. The table attributes should be defined based on the selection.
void TableInsert (HtmlTableProperty tableProperties)	Insert a Table at the current insert point; or over the selected text/control.
bool TableModify (HtmlTableProperty tableProperties)	Modify the Table at the current insert point, or selected table.
void TableInsertRow()	Insert a new row into the table based on the current user row and insertion after.
void TableDeleteRow()	Delete the existing row from the table based on the current user row position.
void GetTableDefinition(out HtmlTableProperty table, out bool tableFound);	Determine if the current insertion point or selected text is contained with a Table element; upon which return the selected table properties.

Enumerations

The following table list the enumerations used to define Html properties:

Name	Description	Values
HtmlListType	Defines the type of list to be inserted.	Ordered Unordered
HtmlHeadingType	Defines the type of Html Heading to mark text selection.	H1 H2 H3 H4 H5
HtmlFontSize	Defines the acceptable values of the size of the FONT.	Default xxSmall xSmall

Name	Description	Values
		Small Medium Large xLarge xxLarge
NavigateActionOption	Defines the navigation window action on a user clicking a Href.	Default NewWindow SameWindow
ImageAlignOption	Defines the image alignment property.	AbsBottom AbsMiddle Baseline Bottom Left Middle Right TextTop Top
HorizontalAlignOption	Define the horizontal alignment property.	Default Left Center Right
VerticalAlignOption	Define the vertical alignment property.	Default Top Bottom
DisplayScrollBarOption	Defines the visibility of the scrollbars.	Yes No Auto
MeasurementOption	Defines the unit of measure for font attributes.	Pixel Percent

Value Types

In support of these two values types there are supporting classes to perform conversion from the enumeration values to and from the Html attribute and style properties.

HtmlFontProperty Struct:

The main purpose of the struct is to be used in replacement of the Framework Font class. The class restricts the acceptable font sizes and extends the attributes.

Property	Type	Description
Name	String	Name of the Font.
Size	HtmlFontSize	Size of the Font.
Bold	Boolean	Bold Indicator.
Italic	Boolean	Italic Indicator.
Underline	Boolean	Underline Indicator.
Strikeout	Boolean	Strikeout Indicator.

Property	Type	Description
Subscript	Boolean	Subscript Indicator.
Superscript	Boolean	Superscript Indicator.

HtmlTableProperty Struct:

The main purpose of the struct is to be used in defining the properties supported for the insertion and maintenance of Html Tables.

Property	Type	Description
CaptionText	String	Caption for the Table.
CaptionAlignment	HorizontalAlignOption	Alignment of the Table caption.
CaptionLocation	VerticalAlignOption	Location of the Table caption; top or bottom.
BorderSize	Byte	Width of the border.
TableAlignment	HorizontalAlignOption	Alignment of the table on the page.
TableRows	Byte	Number of rows in the Table.
TableColumns	Byte	Number of columns in the Table.
TableWidth	UInt16	Width of the Table on the page.
TableWidthMeasurement	MeasurementOption	Measurement option for the Table width; percent or pixels.
CellPadding	Byte	Padding of the Table cells.
CellSpacing	Byte	Spacing of the Table cells.

Events

The following table list the events raised by the Html Editor:

Name	Description	Arguments
HtmlException	Raised when an error is captured from an operation performed internally within the control; such as clicking on a context menu or toolbar. If no event is captured an internal message box should be displayed.	String Operaton Exception ExceptionObject

Visual Elements

Dialogs

The following table lists the dialogs to be presented to the user by the Html Editor. This does not include system defined dialogs; such as the OpenFileDialog and SaveFileDialog:

Name	Caller	Purpose
EditHtml	HtmlContentsEdit	Allow the user to edit the contents of the Body Html.
ViewHtml	HtmlContentsView	Allow the user to view the complete Html markup.
EnterHref	InsertLinkPrompt	Allow the properties of a Href to be defined.
EnterImage	InsertImagePrompt	Allow the properties of an Image to be defined.
FontAttribute	FormatFontAttributesPrompt	Allow the Font properties of the selected text to be defined.
TableProperty	InsertTablePrompt	Allow the properties of a Table to be defined.
FindReplace	FindReplacePrompt	Allow interaction for Find and Replace operations.

Toolbars

The following table lists the operations to be included within a standard toolbar:

Operation	Purpose
TextCut	Cut selected text.
TextCopy	Copy selected text.
TextPaste	Paste from the clipboard.
EditUndo	Undo last operation.
EditRedo	Redo last Undo.
FormatBold	Bold selected text.
FormatUnderline	Underline selected text.
FormatItalic	Italicize selected text.
FormatFontAttributesPrompt	Present user with internal font dialog and mark selected text with the associated attributes.
FormatRemove	Remove font attributes from the selected text.
FormatFontColorPrompt	Present user with color dialog and mark selected text with the associated color.
FormatFontIncrease	Increase the size of the selected text.
FormatFontDecrease	Increase the size of the selected text.
JustifyLeft	Mark selected text as justified Left.
JustifyCenter	Mark selected text as justified Center.

Operation	Purpose
JustifyRight	Mark selected text as justified Right.
FormatTabRight	Indent selected text.
FormatTabLeft	Outdent selected text.
FormatList(HtmlListType.Ordered)	Mark selected text as an ordered list.
FormatList(HtmlListType.Unordered)	Mark selected text as an un-ordered list.
InsertLine	Insert a line over the user's selection.
InsertImagePrompt	Prompt the user for Image properties and insert over the user's selection.
InsertTablePrompt	Prompt the user for Table properties and insert over the user's selection.
InsertHrefPrompt	Prompt the user for Href properties and insert over the user's selection.
FindReplacePrompt	Present the user with a Find and Replace dialog.
DocumentPrint	Print the current document.

Context Menus

Whilst in edit mode the Html editor should replace the standard context menu; with one designed to work with the Html Editor. The new context menu should support all for all the standard text operations, allow for the markup of formatting blocks, and allow for the insert of supported objects (line, image, table, links).

Document specific commands should be available for setting the status of toolbars, scrollbars, and word wrapping. In addition the document commands should support open and saving the Html contents.

Appendix

C# Public API

```
using HtmlDocument = mshtml.HTMLDocument;
using HtmlBody = mshtml.HTMLBody;
using HtmlStyleSheet = mshtml.IHTMLStyleSheet;
using HtmlStyle = mshtml.IHTMLStyle;
using HtmlDomNode = mshtml.IHTMLDOMNode;
using HtmlDomTextNode = mshtml.IHTMLDOMTextNode;
using HtmlTextRange = mshtml.IHTMLTxtRange;
using HtmlSelection = mshtml.IHTMLSelectionObject;
using HtmlControlRange = mshtml.IHTMLControlRange;

using HtmlEventObject = mshtml.IHTMLEventObj;

using HtmlElement = mshtml.IHTMLElement;
using HtmlElementCollection = mshtml.IHTMLElementCollection;
using HtmlControlElement = mshtml.IHTMLControlElement;
using HtmlAnchorElement = mshtml.IHTMLAnchorElement;
using HtmlImageElement = mshtml.IHTMLImgElement;
using HtmlFontElement= mshtml.IHTMLFontElement;
using HtmlLineElement = mshtml.IHTMLHRElement;
using HtmlSpanElement = mshtml.IHTMLSpanFlow;
using HtmlScriptElement = mshtml.IHTMLScriptElement;

using HtmlTable = mshtml.IHTMLTable;
using HtmlTableCaption = mshtml.IHTMLTableCaption;
using HtmlTableRow = mshtml.IHTMLTableRow;
using HtmlTableCell = mshtml.IHTMLTableCell;
using HtmlTableRowMetrics = mshtml.IHTMLTableRowMetrics;
using HtmlTableColumn = mshtml.IHTMLTableCol;


// UserControl class for the HtmlEditor
public sealed class HtmlEditorControl : UserControl
{
      // public event raised if an processing exception is found
      [Category("Exception"),
      Description("An Internal Processing Exception was encountered")]
      public event HtmlExceptionEventHandler HtmlException;

      // public control constructor
      public HtmlEditorControl();

      // create a new focus method that ensure the body gets the focus
      // should be called when text processing command are called
      public new bool Focus();

      // Runtime Display Properties

      // defines the whether scroll bars should be displayed
      [Category("RuntimeDisplay"),
            Description("Controls the Display of Scrolls Bars")]
      [DefaultValue(DisplayScrollBarOption.Auto)]
      public DisplayScrollBarOption ScrollBars;
```

```
// defines the whether words will be auto wrapped
[Category("RuntimeDisplay"),
      Description("Controls the auto wrapping of words")]
[DefaultValue(true)]
public bool AutoWordWrap;

// defines the default action when a user click on a link
[Category("RuntimeDisplay"),
      Description("Window to use when clicking a Href")]
[DefaultValue(NavigateActionOption.NewWindow)]
public NavigateActionOption NavigateAction;

// Defines the editable status of the text
[Category("RuntimeDisplay"),
      Description("Marks the content as ReadOnly")]
[DefaultValue(true)]
public bool ReadOnly;

// defines the visibility of the defined toolbar
[Category("RuntimeDisplay"),
      Description("Marks the toolbar as Visible")]
[DefaultValue(true)]
public bool ToolbarVisible;

// defines the flat style of controls for visual styles
[Category("RuntimeDisplay"),
      Description("Indicates if Control Flat Style is System")]
[DefaultValue(false)]
public bool EnableVisualStyles;

// defines the visibility of the defined toolbar
[Category("RuntimeDisplay"),
      Description("Defines the docking location of the toolbar")]
[DefaultValue(DockStyle.Bottom)]
public DockStyle ToolbarDock;

// Body Properties (Text and HTML)

// defines the base text for the body (design time only value)
// HTML value can be used at runtime
[Category("Textual"),
      Description("Set the initial Body Text")]
[DefaultValue("Html")]
public string InnerText;

// the HTML value for the body contents
// it is this value that gets serialized by the designer
[Category("Textual"),
      Description("The Inner HTML of the contents")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden),Browsable(false)]
public string InnerHtml;

// returns and sets the body tag of the html
// on set the body attributes need to be defined
[Category("Textual"),
      Description("Complete Document including Body Tag")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden), Browsable(false)]
public string BodyHtml;

// return the html tag of the document
// should never be set as contains the HEAD tag
[Category("Textual"),
      Description("Complete Document including Head and Body")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden), Browsable(false)]
public string DocumentHtml;

// Body Properties (Font and Color)

// body background color
// reset and serialize values defined
[Category("Textual"),
      Description("Define the Background Color of the Body")]
public Color BodyBackColor;

// body foreground color
// reset and serialize values defined
[Category("Textual"),
      Description("Define the Foreground Color of the Body")]
public Color BodyForeColor;

// body font definition
// always set based on the controls font
[Category("Textual"),
      Description("Defines the base font name for the text")]
public HtmlFontProperty BodyFont;

// returns or sets the Text selected by the user
[Category("Textual"),
      Description("The Text selected by the User")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden), Browsable(false)]
public string SelectedText;

// returns or sets the Html selected by the user
[Category("Textual"),
       Description("The Text selected by the User")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden), Browsable(false)]
public string SelectedHtml;

// returns any Url that was used to load the current document
[Category("Textual"),
      Description("Url used to load the Document")]
[DesignerSerializationVisibility
      (DesignerSerializationVisibility.Hidden), Browsable(false)]
public string DocumentUrl

// Document Processing Operations

// allow the user to select a file and read the contents
```

```
public void OpenFilePrompt()

// allow the user to persist the Html stream to a file
public void SaveFilePrompt()

// allow the user to load a document by navigation
public void NavigateToUrl(string url);

// allow the user to load a document by url
public void LoadFromUrl(string url);

// allow a user to load a file given a file name
public void LoadFromFile(string filename);

// define the style sheet to be used for editing
// can be used for standard templates
public void LinkStyleSheet(string stylesheetHref);

// return to the user the style sheet href being used
public string GetStyleSheetHref();

// define a script element that is to be used by all documents
// can be sued for document processing
public void LinkScriptSource(string scriptSource);

// return to the user the script block source being used
public string GetScriptBlockSource();

// allow the user to edit the raw HTML
// dialog presented and the body contents set
public void HtmlContentsEdit();

// allow the user to view the html contents
// the complete Html markup is presented
public void HtmlContentsView();

// print the html text using the document print command
// print preview is not supported
public void DocumentPrint();

// toggle the overwrite mode
public void ToggleOverWrite();

// Document Text Operations

// cut the currently selected text to the clipboard
public void TextCut();

// copy the currently selected text to the clipboard
public void TextCopy();

// paste the currently selected text from the clipboard
public void TextPaste();

// delete the currently selected text from the screen
public void TextDelete();
```

```
// select the entire document contents
public void TextSelectAll();

// clear the document selection
public void TextClearSelect();

// undo former commands
public void EditUndo();

// redo former undo
public void EditRedo();

// Selected Text Formatting Operations

// using the document set the font name
public void FormatFontName(string name);

// using the document set the Html font size
public void FormatFontSize(HtmlFontSize size);

// using the document toggles selection with a Bold tag
public void FormatBold();

// using the document toggles selection with a Underline tag
public void FormatUnderline();

// using the document toggles selection with a Italic tag
public void FormatItalic();

// using the document toggles selection with a Subscript tag
public void FormatSubscript();

// using the document toggles selection with a Superscript tag
public void FormatSuperscript();

// using the document toggles selection with a Strikeout tag
public void FormatStrikeout();

// increase the size of the font by 1 point
public void FormatFontIncrease();

// decrease the size of the font by 1 point
public void FormatFontDecrease();

// remove any formatting tags
public void FormatRemove();

// Tab the current line to the right
public void FormatTabRight();

// Tab the current line to the left
public void FormatTabLeft();

// insert a ordered or unordered list
public void FormatList(HtmlListType listtype);

// define the font justification as LEFT
```

```
public void JustifyLeft();

// define the font justification as CENTER
public void JustifyCenter();

// define the font justification as Right
public void JustifyRight();

// Object Insert Operations

// insert a horizontal line in the body
public void InsertLine();

// insert an image tag at the selected location
public void InsertImage(string imageLocation);

// public function to insert a image and prompt user for the link
public void InsertImagePrompt();

// create a web link from the users selected text
public void InsertLink(string href);

// public function to insert a link and prompt user for the href
public void InsertLinkPrompt();

// remove a web link from the users selected text
public void RemoveLink();

// Text Insert Operations

// insert the given HTML into the selected range
public void InsertHtmlPrompt();

// insert the given Text into the selected range
public void InsertTextPrompt();

// returns the acceptable values for the format block
public string[] GetFormatBlockCommands();

// formats the selected text wrapping in the given format tag
public void InsertFormatBlock(string command);

// formats the selected text wrapping in a PRE tag
public void InsertFormattedBlock();

// unformats the selected text removing heading and pre tags
public void InsertNormalBlock();

// inserts a heading tag values Heading 1,2,3,4,5
public void InsertHeading(HtmlHeadingType headingType);

// System Prompt Dialog Functions

// allows the insertion of an image using the system dialogs
public void SystemInsertImage();

// allows the insertion of an href using the system dialogs
```

```
public void SystemInsertLink();

// Font and Color Processing Operations

// using exec command define font properties for selected tag
public void FormatFontAttributes(HtmlFontProperty font);

// using exec command define color properties for selected tag
public void FormatFontColor(Color color);

// display defined font dialog using to set selected text FONT
public void FormatFontAttributesPrompt();

// display system color dialog and use to set selected text
public void FormatFontColorPrompt();

// determine the Font of the selected text range
// set to the default value of not defined
public HtmlFontProperty GetFontAttributes();

// determine if the current font selected is bold given a range
public bool IsFontBold();

// determine if the current font selected is underline
public bool IsFontUnderline();

// determine if the current font selected is italic
public bool IsFontItalic();

// determine if the current font selected is strikeout
public bool IsFontStrikeout();

// determine if the current font selected is Superscript
public bool IsFontSuperscript();

// determine if the current font selected is Subscript
public bool IsFontSubscript();

// Find and Replace Operations

// dialog to allow the user to perform a find and replace
public void FindReplacePrompt();

// reset the find and replace options to initialize a new search
public void FindReplaceReset();

// finds the first occurrence of the given text string
// uses false for the search options
public bool FindFirst(string findText);

// finds the first occurrence of the given text string
public bool FindFirst(string findText,
      bool matchWhole, bool matchCase);

// finds the next occurrence of a given text string
// assumes a previous search was made
// uses false for the search options
```

```
        public bool FindNext(string findText);

        // finds the next occurrence of a given text string
        // assumes a previous search was made
        public bool FindNext(string findText,
              bool matchWhole, bool matchCase);

        // replace the first occurrence of the given string
        // uses false for the search options
        public bool FindReplaceOne(string findText, string replaceText);

        // replace the first occurrence of the given string
        public bool FindReplaceOne(string findText, string replaceText,
              bool matchWhole, bool matchCase);

        // replaces all the occurrences of the given string
        // uses false for the search options
        public int FindReplaceAll(string findText, string replaceText);

        // replaces all the occurrence of the given string
        public int FindReplaceAll(string findText, string replaceText,
              bool matchWhole, bool matchCase);

        // Table Processing Operations

        // present to the user the table properties dialog
        // using all the default properties for the table
        public void TableInsertPrompt();

        // present to the user the table properties dialog
        // ensure a table is selected or insertion point is in table
        public bool TableModifyPrompt();

        // public function to create a table class
        // insert method then works on this table
        public void TableInsert(HtmlTableProperty tableProperties);

        // public function to modify a tables properties
        // ensure a table is selected or insertion point is in table
        public bool TableModify(HtmlTableProperty tableProperties);

        // will insert a new row into the table
        // based on the current user row and insertion after
        public void TableInsertRow();

        // will delete the currently selected row
        // based on the current user row location
        public void TableDeleteRow();

        // based on then user selection return a table definition
        // if table selected (or insertion point in table) return values
        public void GetTableDefinition(out HtmlTableProperty table,
              out bool tableFound);
}


// enum used to insert a list
```

```
public enum HtmlListType
{
      Ordered,
      Unordered
}

// enum used to insert a heading
public enum HtmlHeadingType
{
      H1 = 1,
      H2 = 2,
      H3 = 3,
      H4 = 4,
      H5 = 5
}

// enum used to define the navigate action on a user clicking a href
public enum NavigateActionOption
{
      Default,
      NewWindow,
      SameWindow
}

// enum used to define the image align property
public enum ImageAlignOption
{
      AbsBottom,
      AbsMiddle,
      Baseline,
      Bottom,
      Left,
      Middle,
      Right,
      TextTop,
      Top
}

// enum used to define the horizontal alignment property
public enum HorizontalAlignOption
{
      Default,
      Left,
      Center,
      Right
}

// enum used to define the vertical alignment property
public enum VerticalAlignOption
{
      Default,
      Top,
      Bottom
}

// enum used to define the visibility of the scroll bars
public enum DisplayScrollBarOption
```

```
{
      Yes,
      No,
      Auto
}

// enum used to define the unit of measure for a value
public enum MeasurementOption
{
      Pixel,
      Percent
}


// enum used to modify the font size
public enum HtmlFontSize
{
      Default     = 0,
      xxSmall     = 1,  // 8 points
      xSmall      = 2,  // 10 points
      Small       = 3,  // 12 points
      Medium      = 4,  // 14 points
      Large       = 5,  // 18 points
      xLarge      = 6,  // 24 points
      xxLarge     = 7   // 36 points
}


// struct used for defining FONT attributes (not styles)
[Serializable]
[TypeConverter(typeof(HtmlFontPropertyConverter))]
public struct HtmlFontProperty
{
      // properties defined for the Font
      public string           Name;
      public HtmlFontSize     Size;
      public bool             Bold;
      public bool             Italic;
      public bool             Underline;
      public bool             Strikeout;
      public bool             Subscript;
      public bool             Superscript;

      // constrctor for name only
      public HtmlFontProperty(string name);

      // constrctor for name and size only
      public HtmlFontProperty(string name, HtmlFontSize size);

      // constrctor for all standard attributes
      public HtmlFontProperty(string name, HtmlFontSize size,
            bool bold, bool italic, bool underline);

      // constrctor for all attributes
      public HtmlFontProperty(string name, HtmlFontSize size,
            bool bold, bool italic, bool underline,
            bool strikeout, bool subscript, bool superscript);
```

```
        // constructor given a system Font
        public HtmlFontProperty(System.Drawing.Font font);


        // public method to convert the html into a readable format
        public override string ToString();


        // equals operator for struct comparsion
        public override bool Equals(object fontObject);

        // operator to compare two font attributes for equality
        public static bool operator ==(HtmlFontProperty font1,
              HtmlFontProperty font2);

        // operator to compare two font attributes for inequality
        public static bool operator !=(HtmlFontProperty font1,
              HtmlFontProperty font2);

        // based on a font name being null the font can be assumed Null
        // default constructor will give a null object
        public bool IsNull;
        public bool IsNotNull;

}


// Utility Class to perform Font Attribute conversions
public class HtmlFontConversion
{

        // return the correct string name from a HtmlFontSize
        public static string HtmlFontSizeString(HtmlFontSize fontSize);

        // return the correct bold description for the bold attribute
        public static string HtmlFontBoldString(bool fontBold);

        // return the correct bold description for the bold attribute
        public static string HtmlFontItalicString(bool fontItalic);

        // determine the font size given a selected font in points
        public static HtmlFontSize FontSizeToHtml(float fontSize);

        // determine the font size given the html font size
        public static float FontSizeFromHtml(HtmlFontSize fontSize);

        // determine the font size given the html int size
        public static float FontSizeFromHtml(int fontSize);

        // Used to determine the HtmlFontSize from a style attribute
        public static HtmlFontSize StyleSizeToHtml(string sizeDesc);

        // Used to determine the the style attribute is for Bold
        public static bool IsStyleBold(string style);

        // Used to determine the the style attribute is for Italic
```

```
      public static bool IsStyleItalic(string style);

}

// expandable object converter for the HtmlFontProperty
public class HtmlFontPropertyConverter: ExpandableObjectConverter;


// struct used for defining TABLE attributes
public struct HtmlTableProperty
{
      // properties defined for the table
      public string                 CaptionText;
      public HorizontalAlignOption  CaptionAlignment;
      public VerticalAlignOption    CaptionLocation;
      public byte                   BorderSize;
      public HorizontalAlignOption  TableAlignment;
      public byte                   TableRows;
      public byte                   TableColumns;
      public ushort                 TableWidth;
      public MeasurementOption      TableWidthMeasurement;
      public byte                   CellPadding;
      public byte                   CellSpacing;

      // constructor defining a base table with default attributes
      public HtmlTableProperty(bool htmlDefaults);

}


// Define delegate for raising an editor exception
public delegate void HtmlExceptionEventHandler
      (object sender, HtmlExceptionEventArgs e);


//Exception class for HtmlEditor
public class HtmlEditorException : ApplicationException
{

      // property for the operation name
      public string Operation;

      // Default constructor
      public HtmlEditorException () : base();

      // Constructor accepting a single string message
      public HtmlEditorException (string message)
      : base(message);

      // Constructor accepting a string message
      // and an inner exception
      public HtmlEditorException(string message, Exception inner)
      : base(message, inner);

      // Constructor accepting a single string message
      // and an operation name
      public HtmlEditorException(string message, string operation)
      : base(message);

      // Constructor accepting a string message
      // an operation and an inner exception
      public HtmlEditorException(string message, string operation,
            Exception inner)
      : base(message, inner);

}


// if capturing an exception internally throw an event
// with the following EventArgs
public class HtmlExceptionEventArgs : EventArgs
{

      // constructor for event args
      public HtmlExceptionEventArgs
            (string operation, Exception exception) : base();

      // define operation name property get
      public string Operation;

      // define message exception get
      public Exception ExceptionObject;


}
```

Development Notes

In developing the Html Editor there where some challenges relating to working with the Html DOM and mshtml; these are now discussed.

Initializing the Html DOM

To load the Html Dom within the web browser control one needs to navigate to a document. To initialize the document upon control initialization one can use the "about:blank" document. This contains no text but rather just initialize the Html Dom. It is only after this navigation has been performed that the document and body properties can be examined.

Event Handling

In hooking up to events relating to the Html document and body; a single event handler is used. The prototype for the event method is as such:

```
[DispId(0)]
public void DefaultMethod();
```

Within this method the event object and type can then be examined:

```
HtmlEventObject eventObject = document.parentWindow.@event;
string eventType = eventObject.type;
```

The required events that require capturing are then easily defined by setting the event handler to a class instance (this) of the class containing the DefaultMethod.

Synchronous Navigation

By default when one navigates to a URL, using the documents navigate method, the operation happens asynchronous. Within the content of an Html Editor this behavior is undesirable; as the document cannot be editing until the operation completes. This premise is evident during the control initialization within the designer.

The “about:blank” is navigated to, initializing the Html Dom. After initialization the designer may set persisted properties for the control; some of which may affect the body and hence Html. Thus after the navigation to “about:blank” control can only be passed back to the designer if the Html Dom has been initialized. Similarly if a document is loaded via an URL, the user experience should be that they can immediately start editing the document. Hence only synchronous navigation makes sense.

To perform synchronous navigation once the navigation is performed the code will loop until the DocumentComplete event is triggered; in which a flag is set to indicate the load has completed:

```
loading = true;
this.editorWebBrowser.Navigate(href,
      ref EMPTY_PARAMETER, ref EMPTY_PARAMETER,
      ref EMPTY_PARAMETER, ref EMPTY_PARAMETER);

while (loading)
{
      Application.DoEvents();
      Thread.Sleep(0);
}
```

The DoEvents method has to be called to allow the DocumentComplete event to be process; as it will execute on the same thread as the navigation (STA thread). As the event and navigation execute other thread management classes, like AutoResetEvent, cannot be used.

Href Processing and Html Insertion

One of the side effect of navigating to “about:blank” is that during the insertion of Html any anchor tag may be affected. If the anchor does not include a protocol (base URL) the documents URL, in this case being about:blank, will become the Href attributes prefix. If the original Href attribute is “home.html” it will become “about:blankhome.html”. This undesirable behavior can be remedied by parsing the documents anchors:

```
HtmlElementCollection anchors = body.getElementsByTagName(ANCHOR_TAG);
foreach (HtmlElement element in anchors)
{
      try
      {
            HtmlAnchorElement anchor = (HtmlAnchorElement)element;
            string href = anchor.href;
```

```
                if (href != null &&
                        Regex.IsMatch(href, BLANK_HTML_PAGE,
                        RegexOptions.IgnoreCase)
                        )
                {
                        anchor.href =
                                href.Replace(BLANK_HTML_PAGE, string.Empty);
                }
        }
        catch (Exception)
        {
                // ignore any errors
        }
}
```

This operation occurs during the insertion of Html and replacement of the Body Inner and Outer Html. COM Interop though provides a more elegant solution (see below).

Replacing the Document Body

Most Html elements support two properties for modifying the Html; innerHtml and outerHtml. However the body element does not support the replacement of the outerHtml. To replace the body one has to create a new body element, replace the associated Dom node, and then set the inner html. Thus given a body html one has to first parse this into a body tag and an innerHtml:

```
string bodyElement = string.Empty;
string innerHtml = string.Empty;

if (Regex.IsMatch(value, BODY_PARSE_PRE_EXPRESSION,
        RegexOptions.IgnoreCase | RegexOptions.Multiline |
        RegexOptions.Singleline)
        )
{
        Regex expression = new Regex(BODY_PARSE_EXPRESSION,
                RegexOptions.IgnoreCase | RegexOptions.Multiline |
                RegexOptions.Singleline | RegexOptions.Compiled |
                RegexOptions.ExplicitCapture );
        Match match = expression.Match(value);
        if (match.Success)
        {
                bodyElement = match.Result(BODY_TAG_PARSE_MATCH);
                innerHtml = match.Result(BODY_INNER_PARSE_MATCH);
        }
}
if (bodyElement == string.Empty)
{
        bodyElement = BODY_DEFAULT_TAG;
        innerHtml = value.Trim();
}

HtmlDomNode oldBodyNode = (HtmlDomNode)document.body;
HtmlDomNode newBodyNode =
        (HtmlDomNode)document.createElement(bodyElement);
oldBodyNode.replaceNode(newBodyNode);

body = (HtmlBody)document.body;
```

```
body.innerHTML = innerHtml;
```

To perform the parsing of the Html a regular expression is used. The bodyElement is defined as the open and close body tags, the innerHtml being the internal contents of this tag.

Font Names and Format Blocks

During the display of the font dialog, the names of available fonts is derived from the system installed font which support regular, bold, italic, and underline styles.

During the display of the format block commands the list of available options is hardcoded as “Formatted”, “Heading 1”, “Heading 2”, “Heading 3”, “Heading 4”, “Heading 5”, and “Normal”.

There is a class called BlockFormatsClass and FontNamesClass but they are not easily instantiated. If one creates a new HtmlDlgSafeHelperClass, the return for the formats and fonts is always null.

Error Handling

There are two aspects to error handling within the Html Editor:

- Standardizing internal error with the creation of a new exception type.
- Capturing exceptions that arise from events generated from within the control; such as those generated when the user interacts with the internal toolbar and context menu.

Within the application there is an exception type, defined as HtmlEditorException, which is derived from ApplicationException. This exception type extends the base type with the inclusion of an operation; being defined as a string name to represent the html editing operation taking place.

With the inclusion of a toolbar and context menu within the Html Editor Control there is the possibility that errors can be generated from user interaction that the hosting container knows nothing about. Thus all events toolbar and menu events within the control are encapsulated with a try block.
Any caught error is processed in one of two ways:

- If the container has defined a delegate to capture the HtmlException event, the delegate is called. This allows the container to define the action in the advent of an internal processing error.
- If the container has defined a delegate to capture the HtmlException event, a dialog is presented informing the user of the error. As the exception was caused by a user interaction a dialog is an appropriate action.

Allowing the container application to handle the internal exception allows for a consistent presentation of application error rather than having to use the control defined dialog.

COM Interop

COM Interop becomes necessary; in preventing Url's from being modified on paste operations. There is a Command called No Fixup Url's on Paste that needs to be executed.

No Fixup Urls on Paste

The HtmlDocument implements the interface IOleCommandTarget, from which the Exec method provides the ability to execute commands within a group.
By default when Url's are pasted into an Html Document relative Url's are modified to be fully qualified with the documents Base Url. In the advent that this Base Url is the "about:blank" reference this behavior is someone undesirable. It becomes even more apparent when working with bookmarks.

To remedy his situation the Html Document can be cast to the IOleCommandTarget, and the command IDM_NOFIXUPURLSONPASTE can be executed:

```
IOleCommandTarget target = null;
int hResult = COM.HRESULT.S_OK;
try
{
      target = (IOleCommandTarget)document;
      hResult = target.Exec(ref CommandGroup.CGID_MSHTML,
            (int)CommandId.IDM_NOFIXUPURLSONPASTE,
            (int)CommandOption.OLECMDEXECOPT_DONTPROMPTUSER,
            ref EMPTY_PARAMETER, ref EMPTY_PARAMETER);
}
catch (Exception ex)
{
      hResult = Marshal.GetHRForException(ex);
}
if (hResult == COM.HRESULT.S_OK)
{
      rebaseUrlsNeeded = false;
}
else
{
      rebaseUrlsNeeded = true;
}
```

This command ensures that any pasted Url's remain unchanged during paste operations.