Specifications for Graphite-enabled Edit Control

Sharon Correll,
SIL Non-Roman Script Initiative (NRSI)
2003-04-03

1. Introduction

This document describes the specifications for the Graphite-enabled edit control that is being developed by SIL International in accordance with their contract with UNESCO.

2. Phases of development

The control will be developed in a series of phases, as follows:

- Phase 1: Single-line control with basic editing capabilities; cut/copy/paste; single default font and size; ability to read and save plain text to a string.
- Phase 2: Multi-line control with line wrapping; multiple paragraphs.
- Phase 3: Mixture of fonts and sizes; read and save using HTML format.
- Phase 4: Event handling for key events and selection change; text insertion and manipulation via TOM; search and replace; undo and redo.
- Phase 5: Right-to-left layout and related features; ensure proper behavior of Unicode supplementary-plane characters.
- Phase 6: Character and paragraph formatting; styles.
- Phase 7: Support of large files.
- Phase 8: Support of additional formats (e.g., RTF).
- Phase 9: Additional multilingual behavior.
- Phase 10: Implementation of COM interfaces to support standard use in a web browser.

3. Architecture

The edit control will consist of a DLL for an ActiveX control that can be embedded within a host application. The control will run on Windows 2000 and XP operating systems.

3.1. COM interface support

3.1.1. Phase 1

The first phase of development will support a simple COM interface that is adequate to initialize the control with data.

Properties must be set before the control is opened. Required properties are:

- DefaultFont(char * fontname)—the default font of the control; if not set, the default is Arial.
- DefaultFontSize(int fontsize)—the default font size of the control, in points; if not set, the default is 10 points.
- Text(BSTR)—the text that is in the control; plain text format. (Alternately, could use PutText and GetText methods with OLECHAR * and character-count arguments.)

Methods:

- Copy()—copies the selected text into the clipboard.

- Cut()—copies the selected text into the clipboard and deletes it from the control.
- Paste()—pastes the text from the clipboard into the control.

Also the standard ActiveX control mechanism will be used to set the physical height and width of the control.

3.1.2. Phase 2

Additional property:

- MultiLine(bool)—tells the control whether it should function as a single-line or multi-line editor; if not set, the default is false.
- VerticalScroll(bool)—tells the control whether to include a vertical scroll bar. Will have no effect if the control is not multi-line.
- HorizontalScroll(bool)—tells the control whether to include a horizontal scroll bar. Will have no effect if the control is not multi-line.

3.1.3. Phase 3

Additional property:

- FontAndSizeDialog(bool)—tells the control whether it should include the option to launch a font-and-size dialog on its right-click menu. [This is an optional feature; see section 6.3.3.]

Additional methods:

- GetSelectionFont(BSTR *)—returns the name of the font that is used for the selection. If there is no selection, or the selection uses a mixture of different fonts, the empty string will be returned.
- GetSelectionFontSize(int * pfontsize)—returns the font size, in points, that is used for the selection. If there is no selection, or the selection uses a mixture of the different font sizes, zero will be returned.
- SetSelectionFont(char * fontname)—changes the font of the selected range of characters. If the selection is an insertion point, immediate subsequent typing will use the given font. (If the insertion point is moved, however, typing will use the given font of the new selection.)
- SetSelectionFontSize(int fontsize)—changes the font size of the given range of characters. The font size is in points. If the selection is an insertion point, subsequent typing will use the given font size.
- PutHtmlText(BSTR)—puts text into the control using HTML format. (Possibly using OLECHAR * and length arguments would be acceptable in place of a BSTR.)
- GetHtmlText(BSTR *)—returns the text that is in the control in HTML format. (Possibly using OLECHAR *, buffer length, and return length arguments would be acceptable in place of a BSTR.)

Later versions of the control will support standard COM interfaces ITextDocument and ITextRange. Later phases will require support of ITextSelection, ITextFont, and ITextPara. The following methods will be implemented:

3.1.4. Phase 4

Phase 4 involves adding the Text Object Model interfaces to the control. This will involve implementing the methods named below. The behavior of these methods is specified in Microsoft's MSDN documentation.

3.1.4.1. ITextDocument

- Freeze
- GetSaved
- GetSelection
- Range
- RangeFromPoint
- Redo
- SetSaved
- Undo
- Unfreeze

3.1.4.2. ITextRange

- CanEdit
- CanPaste
- Collapse
- Copy
- Cut
- Delete
- EndOf
- Expand
- FindText
- FindTextEnd
- FindTextStart
- GetChar
- GetDuplicate
- GetEnd
- GetFont
- GetFormattedText
- GetIndex
- GetPara
- GetPoint
- GetStart
- GetStoryLength
- GetStoryType
- GetText

- InRange
- IsEqual
- Move
- MoveEnd
- MoveEndUntil
- MoveEndWhile
- MoveStart
- MoveStartUntil
- MoveStartWhile
- MoveUntil
- MoveWhile
- Paste
- ScrollIntoView
- Select
- SetChar
- SetEnd
- SetFont
- SetFormattedText
- SetIndex
- SetPara
- SetPoint
- SetRange
- SetStart
- SetText
- StartOf

3.1.4.3. ITextSelection

- GetFlags
- GetType
- SetFlags
- TypeText

3.1.5. Phase 6

Support of character and paragraph styles will require implementation of ITextFont and ITextPara.

3.1.5.1. ITextFont

- CanChange
- GetAllCaps
- GetBackColor
- GetBold
- GetDuplicate
- GetForeColor
- GetHidden
- GetItalic
- GetLanguageID
- GetName
- GetOutline
- GetName
- GetOutline
- GetPosition
- GetProtected
- GetSize
- GetSmallCaps
- GetSpacing
- GetStrikeThrough
- GetStyle
- GetSubscript
- GetSuperscript
- GetUnderline
- GetWeight
- IsEqual
- Reset
- SetAllCaps
- SetAnimation
- SetBackColor
- SetBold
- SetDuplicate
- SetForeColor

- SetHidden
- SetItalic
- SetLanguageID
- SetName
- SetOutline
- SetPosition
- SetProtected
- SetShadow
- SetSize
- SetSmallCaps
- SetSpacing
- SetStrikeThrough
- SetStyle
- SetSubscript
- SetSuperscript
- SetUnderline
- SetWeight

3.1.5.2. ITextPara

- AddTab
- CanChange
- ClearAllTabs
- DeleteTab
- GetAlignment
- GetDuplicate
- GetFirstLineIndent
- GetLeftIndent
- GetLineSpacing
- GetRightIndent
- GetSpaceAfter
- GetSpaceBefore
- GetStyle
- GetTab
- GetTabCount

- IsEqual
- Reset
- SetAlignment
- SetDuplicate
- SetIndents
- SetLeftIndent
- SetLineSpacing
- SetRightIndent
- SetSpaceAfter
- SetSpaceBefore
- SetStyle

3.1.6. Phase 8

- ITextDocument:Open—extended to handle additional formats, such as RTF.
- ITextDocument:Save—extended to handle additional formats, such as RTF.

4. XHTML format

The control will support the following subset of the XHTML format described in the document "XHTML Format for WorldPad File Equivalence".

4.1. Phase 3

Tags: (literal:html), (literal:body), (literal:p), (literal:span).

Style properties: (literal:font-family), (literal:font-size) (both absolute and relative sizes).

4.2. Phase 5

Style properties: (literal:direction) (e.g., (literal:style="direction: rtl")).

4.3. Phase 6

Style properties: (literal:style), (literal:font-style), (literal:font-weight), (literal:color), (literal:background-color) [optional, since it is not required by Paratext], (literal:vertical-align), (literal:text-decoration), (literal:text-align), (literal:text-indent), (literal:margin-left/right/top/bottom), (literal:line-height) (single, 1_, double spacing).

5. Layout capabilities

5.1. Single-line control [Phase 1]

For a single-line edit control, the entire text will be displayed on a single line that scrolls horizontally. Typing off the trailing side (i.e., in a left-to-right control, the left side) of the control will cause the control to scroll horizontally; similarly deleting or moving the selection with the arrow keys will scroll to keep the selection visible. No scroll bar is required.

Insertion of paragraph-breaking characters such as the Return key will be ignored. Pasting data or reading data from a file that includes paragraph-breaking characters will have the effect of inserting just the first line of the text.

For phase 1, the single-line edit control will be able to handle approximately 256 characters of data with good performance. Later phases will signficantly raise that limit or remove it altogether.

5.2. Multi-line control [Phase 2]

A multi-line edit control will provide automatic line wrapping for text in a single paragraph. The wrapping behavior will be based on the behavior of the Graphite font or the white-space character properties of data that is not rendered with Graphite. A vertical and/or horizontal scroll bar will be available, if requested.

Paragraph breaks will occur wherever there is paragraph formatting in the HTML-based text or return characters in plain text. In plain text, the characters or character sequences listed in the table in section 5.1 will cause paragraph breaks. A small amount of vertical leading will be used to visually separate paragraphs. Paragraph-level formatting such as line spacing and indentation are not required.

A multi-line edit control will be able to handle approximately 2000 characters of data with good performance at phase 2. Later phases will require good performance for many pages of text.

5.3. Multilingual layout support [Phase 5]

Both the single- and multi-line controls will be capable of right-to-left (as well as left-to-right) layout. The layout direction is specified by a parameter passed from the host application. Mixing of left-to-right and right-to-left paragraphs in a single control is not required.

Both forms of the control are likewise required to handle a mixture of fonts, some of which may require right-to-left directionality and others left-to-right. The multi-line control should correctly handle the problem trailing white-space in mixed-direction paragraphs according to the Unicode bidirectional algorithm (i.e., trailing white space in an “upstream” run of text is treated as if it were “downstream” and thus moved to the end of the line).

6. Editing capabilities

6.1. Typing [Phase 1]

In a multi-line control, the Return key will produce a paragraph break. In a single-line control, the Return key will be ignored.

When a range selection exists, the Delete key will delete the range of text and create an insertion-point selection at the location of the deletion. When an insertion point exists, the Delete key will delete the character that is (logically) ahead of the insertion point. In either case the resulting insertion-point will have the properties of the (first) deleted character. Note that when working with supplementary-plane Unicode characters, a character may consist of two 16-bit items. [Phase 5]

When a range selection exists, the Backspace key will delete the range of text and create an insertion-point selection at the location of the deletion. When an insertion point exists, the Backspace key will delete the character that is (logically) before the insertion point. In either case, the resulting insertion-point will have the properties of the (first) deleted character. Changing the insertion mode using the Insert key is not required.

6.2. Creating and manipulating selections [Phase 1]

6.2.1. Clicking the mouse

A single mouse click will produce an insertion point at the closest location between two characters or at the beginning or end of the text. The insertion point will have the properties (i.e., subsequent typing will produce characters with the properties) of the closest character receiving the click.

6.2.2. Dragging

Dragging the mouse will produce a range selection extending to the point where the mouse is released.

More sophisticated drag features such as word-level selection are not required.

Drag-and-drop editing of text is low priority.

6.2.3. Shift-clicking

Shift-clicking will produce a range selection extending from the previous insertion-point or range anchor to the point where the mouse is released.

Whole-word selection when shift-clicking is not required.

6.2.4. Arrow keys

There are two possible modes of behavior for arrow keys.

6.2.4.1. Logical mode [Phase 1]

In a left-to-right control, the right arrow key will move an insertion point forward in the data to the next legitimate insertion point. When starting from a range selection, the right arrow key will reduce the selection to the trailing end of the range.

Similarly, the left arrow key will move an insertion point backward in the data to the next legitimate insertion point. When starting from a range selection, the left arrow key will reduce the selection to the leading end of the range.

The directions are reversed in a right-to-left control [Phase 5]. The right arrow key will move an insertion point backward, or reduce a range to the leading end. The left arrow key will move an insertion point forward, or reduce a range to the trailing end.

The shifted arrow keys extend the selection to the location that would be the result of pressing the normal arrow key. When starting from a range selection, it is the current end-point of the range that is used to determine the next end-point.

6.2.4.2. Visual mode [Phase 9]

In visual mode, the left arrow key moves an insertion point selection to the next viable position to the left, and the right arrow key moves it similarly to the right. This requires interaction with the rendering engine which knows the physical arrangement of the glyphs.

When starting from a range selection, the right arrow key behaves identically to logical mode.

The shifted arrow keys extend the selection to the location that would be the result of pressed the normal arrow key.

6.2.5. Home and End keys

6.2.5.1. Logical mode [Phase 2]

The Home key moves the selection to just before the (logically) first character on the line, and the End key moves the selection to just after the (logically) last character on the line.

The Shift-Home key extends the selection from the current anchor point to just before the first character on the line, and the Shift-End key extends the selection to just after the last character on the line. When starting from a range selection, the current line is considered to be the one containing the end-point of the range, not the anchor.

6.2.5.2. Visual mode [Phase 9]

In visual mode, in a left-to-right paragraph, the Home key moves the selection to the left-most viable location on the current line, and the End key moves the selection to the right-most viable location. In a right-to-left paragraph, the Home key moves the selection to the right-most location, and the End key moves the selection to the left-most location.

The Shift-Home key extends the selection from the current anchor point to the left-most viable location on the current line (or right-most, in a right-to-left paragraph). The Shift-End key extends the selection from the current anchor point to the right-most viable location on the current line (or left-most, in a right-to-left paragraph). When starting from a range selection, the current line is considered to be the one containing the end-point of the range, not the anchor.

6.2.6. Double-clicking

Double-clicking will create a range selection covering a single word of text.

Determining correct word boundaries for scripts that do not use white space between words is not required.

Handling triple-clicking, e.g., to select a paragraph, is not required.

6.3. Formatting [Phase 3]

For phases 3 – 5, the only formatting options that are required relate to font and size.

6.3.1. Font

The control will be initialized with a default font, which will be used for all plain text and any formatted text where font is unspecified. For text specified in HTML, the font will be indicated by the font-family style property as discussed in the HTML specification.

6.3.2. Size

The control will be initialized with a default font, which will be used for all plain text and any formatted text where font is unspecified. For text specified in HTML, the font will be indicated by the font-size style property as discussed in the HTML specification.

Font size can be indicated in either absolute or relative terms. Absolute terms means the exact point size is indicated, such as 18 points. Relative means a percentage of the default font size, such as 150% for a font that is larger than the default font size of the control.

6.3.3. UI for setting font and size

The edit control may include a right-click menu option to allowing the setting of font and size. Choosing this item brings up a small dialog containing font and size combo-boxes. [This is an optional feature.]

7. Input and rendering capabilities

7.1. Keyboard input

The control will be compiled as a Unicode window, which means that data received from the keyboard will be 16-bit Unicode characters. It is expected that various system keyboards as well as Keyman keyboards will be used to generate input. However, it is not required to programmatically or automatically choose any keyboard, for instance, as a result of clicking, using arrow keys, or changing the font.

7.2. Rendering

The control will be able to render using the Graphite engine as well as generate simple rendering needed for basic Roman text. (Good support for Uniscribe rendering is available using the same interface as the Graphite engine.)

The Graphite engine will be used for any range of text that uses a Graphite-enabled font.

Behaviors required for complex scripts, such as split cursors and bidirectionality, will be provided by using the Graphite engine.

END