# Formatted Text Control

User Guide - Version 3.2.0

**BKeeney Software inc.**

Revised: February 2019

# Table of Contents

# Welcome to the Formatted Text Control

The Formatted Text Control (FTC) is a canvas based control for Xojo that implements word processor capabilities similar to Apple's Pages or Microsoft's Word. The FTC is useful for implementing reports or general word processing capabilities within your applications. The FTC supports Mac OS X and Microsoft Windows. The software is completely written in Xojo for maximum compatibility and flexibility.

### Minimum Requirements

The FTC version 3.2 supports Xojo Release 2016 R4 and above (for now). It will NOT work on Real Studio - period. The FTC is officially supported for OS X Cocoa and Windows. It may work in Linux in limited fashion but we do not officially support Linux.

### Text Input Canvas Plugin

Starting with version 3.1 the FTC requires the use of the TextInputCanvas plugin. The plugin is an open source plugin published by Xojo Inc. that allows Cocoa applications to receive keydown and keyboard handling events. The plugin is Windows and Linux compatible so there is little to no changes between the platforms.

The plugin is part of the FTC distribution package, but it can be found at https://github.com/xojo/TextInputCanvas. It may also be found in the Required Plugin folder of the download package.

### Support

If you need support or find a defect, send email to our support email at...

support@bkeeney.com

Visit our website www.bkeeney.com for updates.

## FTC Bug Reporting

To report bugs or request new or changed functionality, please report them in our Mantis Bug Tracking software. You can request an account and log in at https://mantis.bkeeney.com

## Installing the Software

Installing the software into your project is easy to do!

1) Copy the “FTC” folder into your project from the distribution project (i.e. from project to project).
2) Drag and drop the FormattedText class to the window or create a Canvas control and change the super class of the control to “FormattedText”.
3) Drag and drop a couple of FTScrollBar classes to the window or create two Scrollbar controls and place them on top of the FTC control and then change the super class of these two scrollbars to “FTScrollBar”.
4) Fill in the events “GetHorizontalScrollbar” and “GetVerticalScrollbar” in the FTC to inform the FTC to use the scrollbars created in the previous step.
5) Set the design time properties in the FormattedText control and in the scrollbars. For example, in the scrollbars set the LiveScrolling property to true to turn on live scrolling.
6) Create an “EditRedo” menu item in your menubar.
7) In the window’s “Resized” and “Resizing” events call the FTC “resize” method according to your application’s needs.

You are now ready to use the FTC!

# Demo Application

The demo application provided with the source code serves two purposes. First it serves as a test bed for the FTC functionality. The second purpose is to show you how to access the FTC functionality in a working application. Sometimes when working with complex software like the FTC, there is no substitute for a working example to show you how to do something. The demo application is meant to be a starting point and not the prescribed way you should build the GUI for your application. The way the functionality is laid out in the demo application is designed to show the functionality that is within the FTC. You can use as little or as much of the FTC functionality as you desire.

To run the demo you will the need the following plugins.

- BKS Spell Checker Plugin

  http://www.bkeeney.com/allproducts/bks-spell-checker-plugin-for-xojo/

- MBS REALbasic ComputerControl Plugin.rbx

  http://www.monkeybreadsoftware.de/realbasic/plugins.shtml

The main demo application no longer has spell checking enabled so that you don't need the BKS Spell Checker plugin installed.  Check out the specific Spell Checking demo project to see how it is used.  If you wish to use the BKeeney Software spell checking plugin your applications, it can be purchased separately.  If you wish to turn the spell checking on set the

The MBS plugin is required for doing regression testing within the demo. You can eliminate the use of the MBS plugin by setting the constant "FTC_REGRESSION_TEST" to false in the demo project.

There is one spot in FTUtilities that uses the MBS plugin to determine if a file is an image file.  We have turned this off by default, if you wish to use it, set the "MBS_INSTALLED" constant to true.

# Customizing the FTC

There are several ways to customize the FTC to fit your needs. You can change the controls design time properties, fill in the event methods, create subclasses, use the built in tags, and add custom objects. If you intend to reuse the settings and event code, then it is probably best to create a subclass.

Throughout the FTC, the get/set/find paradigm is used to access the data. These methods provide a level of indirection that allows the FTC to be developed independent of your subclasses. By convention, get and set methods are computationally inexpensive, but the find methods are relative expensive because they must go out and search the internal data structures. The get/set/find convention also allows you to

quickly find the appropriate method as you type in code and use the auto completion functionality of Xojo.

Note, all values (margins etc.) are internally measured in inches.

**Design Time Properties**

*AllowBackgroundUpdates As boolean*

Allow background processing to be performed. For example, when a document is loaded with pictures, the RTF representation is calculated in the background in order to save time when the you save the content as RTF.

*AllowPictures As boolean = true*

Allow pictures to be loaded into the FTC. If you want the FTC to behave more like an TextArea, then you can use this property to turn off the importation of pictures.

*BackgroundColor As Color*

The color that is displayed behind the pages in the control.

*BorderColor As color*

The color of the border surrounding the control.

*BottomMargin As double*

The relative margin from the bottom of a page. To specify a 1 inch margin at the bottom of the page set this property to 1.0.

*CaretColor As Color*

The color of the insertion caret.

*DefaultFont As string*

The font to be used when creating a style run when no font is specified.

*DefaultFontSize As integer*

The font size to be used when creating a style run when no font size is specified.

*DefaultTabStop As double*

The default tab stop in inches.

*DefaultTextColor As Color*

The default color of text.

*DragAndDrop As boolean*

Allow drag and drop operations to be performed.

*DrawControlBorder as boolean*

Enables or disables the drawing of the border around the control.

*EditViewMargin As double*

When the control is in Edit mode, this property specifies the margin on all sides of the text in inches.

*EditViewPrintMargin*

When the control is in Edit mode, this property specifies the margin on all sides of the text in inches when the document is printed.

*HScrollbarAutoHide As boolean*

When set to true, will cause the horizontal scrollbar to hide when the length of the content can be fully displayed without scrolling.

*InhibitSelections As boolean*

This property controls the ability to allow selections.

*InvisiblesColor As Color*

The color of invisible characters such as the tab or pilcrow.

*LeftMargin As double*

The relative margin from the left side of a page. To specify a 1 inch margin at the left side of the page set this property to 1.0.

*MarginGuideColor As Color*

The color to use when displaying margin guides.

*Mode As integer*

Set the initial mode of the control which can be Page, Normal, Edit, or Single mode.

*PageHeight As double*

Set the initial page height.

*PageWidth As double*

Set the initial page width.

*ReadOnly As boolean*

Restrict the control to a read only status so the user can't change the content.

*RightMargin As double*

The relative margin from the right side of a page. To specify a 1 inch margin at the right side of the page set this property to 1.0.

*ShowInvisibles As boolean*

This property controls the visibility of markers which are normally invisible.

*ShowMarginGuides As boolean*

Controls whether or not margin guides are shown when the control is opened.

*ShowPilcrows as boolean*

The pilcrow character is the backwards "P" shown at the end of a paragraph. This property allows you control the display of these markers.

*SpellCheckWhileTyping as boolean*

This property allows you to turn on spell checking as you type functionality. See the spell checking section in this manual to see what else you have to do to implement spell checking.

*TopMargin As double*

The relative margin from the top side of a page. To specify a 1 inch margin at the left side of the page set this property to 1.0.

*UndoLimit As Integer*

The number of undo actions that will be tracked by the control. If you set the value to -1 or less, the undo limit is unbounded (meaning no limit to the number of undo actions). It is not recommended to have an unbounded undo stack since it will grow without bound.

*VScrollbarAutoHide As boolean*

When set to true, will cause the vertical scrollbar to hide when the length of the content can be fully displayed without scrolling.

### FormattedText Events

These events are the similar in meaning as you would find in any other Xojo control.

*Activate()*

The control is being activated.

*BaselineAtIndex(index as integer)*

Used by the TextInputCanvas plugin to get the baseline position of the index passed in.

*CharacterAtPoint(x as integer, y as integer) as integer*

Used by the TextInputCanvas plugin to get the offset of the character given the x and y location passed in.

*Close()*

The control is closing.

*ConstructContextualMenu(base as MenuItem, x as integer, y as integer) as boolean*

This is where you construct your contextual menu for the control.

*ContextualMenuAction(hitItem as MenuItem) as boolean*

An item has been chosen from the contextual menu. Return true if you handled it.

*Deactivate()*

The control is being deactivated.

*DefaultParagraphStyle() as FTParagraphStyle*

You can use this event to specify exactly how you want new paragraphs to look and feel. Create a new FTParagraphStyle object and then fill it in with the desired attribute values and then return it.

*DiscardIncompleteText()*

Used by the TextInputCanvas plugin to tell the control to disregard any existing incomplete text.

*DoCommand(command as string)*

Used by the TextInputCanvas plugin to pass in commands to the control. This is where a lot of work takes place not directly related to text input. For example, if the user presses the Home key the CmdScrollToBeginningOfDocument is sent. It is up to the control to then handle it.

*DoubleClick(x as integer, y as integer)*

A double click event has occurred.

*DragEnter(obj as DragItem, action as integer) as boolean*

The mouse has entered the control during a drag event.

*DragExit(obj as DragItem, action as integer)*

The mouse has left the control during a drag event.

*DrawLine(proxy as FTParagraphProxy, line as FTLine, g as graphics, drawBeforeSpace as boolean, firstIndent as integer, page as integer, x as double, y as double, leftPoint as integer, rightPoint as integer, alignment as integer, printing as boolean, printScale as double)*

Allows you to draw on a composed line basis. For example, add line numbers to the display.

*DragOver(x as integer, y as integer, obj as DragItem, action as integer) as boolean*

The is being moved within the control during a drag event.

*DropFolderItem(fi as FolderItem, offset as FTInsertionOffset)*

A folder item has been dropped on the control at the specified insertion offset.

*DropObject(obj as DragItem, action as integer) as boolean*

An object has been dropped on the control. Return true if you handled the drop.

*EnableMenuItems()*

Enable the menu items when the control has the focus.

*FontNameAtLocation(location as integer) as String*

Used by the TextInputCanvas plugin to get the name of the font at the passed in location.

*FontSizeAtLocation(location as integer) as Integer*

Used by the TextInputCanvas plugin to get the font size at the passed in location.

*GetHorizontalScrollbar() as FTScrollbar*

Tell the control to use this scrollbar for horizontal scrolling. You are not required to have a horizontal scrollbar.

*GetVerticalScrollbar() as FTScrollbar*

Tell the control to use this scrollbar for vertical scrolling. You are not required to have a vertical scrollbar.

*GotFocus()*

The control has received the focus.

*HyperlinkClick(sURL as string)*

A hyperlink has been clicked in the control. It is up to you to do something with it such as use the REALbasic ShowURL method to perhaps open the default web browser with the passed in URL.

*IncompleteTextRange() as TextRange*

Used by the TextInputCanvas plugin to get the text range of the current incomplete text range (if there is one).

*InsertImage(pic as FTPicture) as boolean*

A picture is being inserted. Use this event to allow or disallow the insertion and handle the picture as you see fit. Return true if you handled the picture.

*InsertionPointChanged(previousOffset as FTInsertionOffset, currentOffset as FTInsertionOffset)*

The insertion caret position has changed.

*InsertText(text as string, range as TextRange)*

Used by the TextInputCanvas plugin to pass in text to put in the control in a certain range.

*IsEditable() as boolean*

Used by the TextInputCanvas plugin to determine if the text in the control is editable. Currently we return true by default since the control determines what changes in the control.

*LostFocus()*

The control is losing the focus.

*MouseDown(x as integer, y as integer) as boolean*

A mouse down event has occurred. Return true if you handled the event and you don't want further processing.

*MouseDrag(x as integer, y as integer)*

A mouse drag event has occurred.

*MouseEnter()*

The mouse cursor has crossed over into the control.

*MouseExit()*

The mouse cursor has crossed out of the control.

*MouseMove(x as integer, y as integer)*

The mouse cursor has moved.

*MouseUp(x as integer, y as integer)*

A mouse up event has occurred in the control.

*MouseWheel(x as integer, y as integer, deltaX as integer, deltaY as integer) as Boolean*

A mouse wheel event has occurred in the control. Return true if you handled the event.

*NewDocument() as FTDocument*

If you wish to create a custom subclass of the FTDocument class, you instantiate it here and return it. This event is only called once when the control is opened.

*NewPage() as FTPage*

If you wish to create a custom subclass of the FTPage class, you instantiate it here and return it. This event is call every time the FTC needs a new page object.

*ObjectDoubleClick(*obj as FTObject*)*

If you wish to handle double click events on any FTCustom objects that have been placed in your document this event is called. It is up to you to then deal with it. FTCustom subclasses that override the MouseDown event may have to call this even on their own.

*Open()*

The control is opening.

*Paint(g as Graphics)*

The control needs to be refreshed.

*PasteAction as Boolean*

This event will allow you to handle the paste event. For example, perhaps you want to strip off any formatting and just paste plain text. If you return false from this event, the built in functionality will be done. If you return true, it means you handled it. In either case the undo functionality is handled for you.

*PostDisplayDraw(g as graphics)*

This event gives you access to the display buffer just after the FTC completes drawing its contents and before it blits it to the screen. This event allows you draw any additional elements on the face of the control.

*PostPageDraw(g as graphics, p as FTPage, printing as boolean, printScale as double)*

After the FTC has drawn all the content on a page, this event is called so you can perform any drawing operations. You could use this event to draw headers or page numbers.

*PrePageDraw(g as graphics, p as FTPage, printing as boolean, printScale as double)*

Before the FTC has drawn any content on a page, this event is called so you can perform any drawing operations. You could use this event to draw a watermark underneath the text on the page.

*RectForRange(byref range as TextRange) as REALbasic.Rect*

Used by the TextInputCanvas plugin to get the rectangle to cover the passed in range.

*ScrollChanged(verticalPosition as integer, horizontalPosition as integer)*

The user has changed the scrollbar.

*SelectedRange() as TextRange*

Used by the TextInputCanvas plugin to the range of currently selected text.

*SelectionChanged()*

The selection inside the control has changed.

*SetIncompleteText(text as string, replacementRange as TextRange, relativeSelection as TextRange)*

Used by the TextInputCanvas plugin to tell the FTC to insert text that might be incomplete. For example, some keyboard combinations (mainly on Mac OS X) require two actions such at ü which requires the user to first press option u to get the umlaut (the incomplete text) and finally the 'u' which completes it.

*SingleClick(x as integer, y as integer)*

A single click has occurred in the control.

*SpellCheckParagraph(text as string) as FTSpellCheckWord()*

The background spell checking thread calls this event for every paragraph it is spell checking. You return an array of FTSpellCheckWord objects that indicate the words you marked as misspelled. See the demo application TestWindow for an example of how to implement this method.

*SpellCheckWord(word as string) as boolean*

When spell checking while typing is active, the FTC will parse words and send to this event to be spell checked. Return true if the word is spelled correctly.

*TextChanged()*

The content of the control has changed. Be careful what you implement in this event since it will affect typing speed which is directly related to the perceived speed of the control.

*TextForRange(range as TextRange) as string*

Used by the TextInputCanvas plugin to get the text for the passed in TextRange.

*TextLength() as integer*

Used by the TextInputCanvas plugin to get the length of the text in the current document.

*TripleClick(x as integer, y as integer)*

A triple click has occurred in the control.

*UnrecognizedXmlObject(obj as XmlElement)*

The object was not handled by the FTC or within the XmlObjectCreate event.

*XmlEncodeTag(obj as FTBase) as string*

Return an encoded string representing the obj.tag field. See the demo application for an example on how to code this up.

*XmlDecodeTag(data as string, obj as FTBase)*

Take the data passed in and decode it and assign it to the obj.tag field. See the demo application for an example on how to code this up.

*XmlObjectCreate(obj as XmlElement) as FTObject*

While parsing the XML input stream, the control encountered an unknown element which is probably a custom control. You need to query the object and create the proper control and then return it. The FTC will then insert it into the document.

**FTDocument Events**

*EndXMLLoad()*

The XML data has finished loading and this event gives you a chance to perform some post processing.

*LoadXMLItem(item as XMLElement)*

An unknown item was encountered in the XML data stream. These are items that are children of the root node that were added through the SaveXML event.

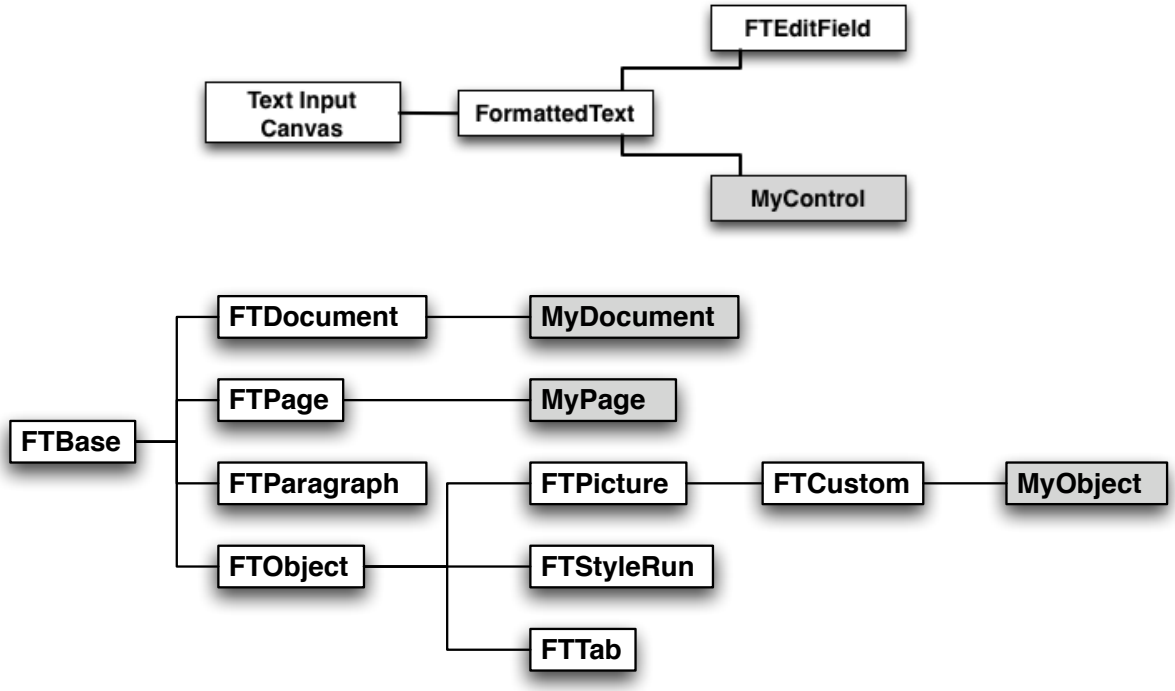
*SaveXML(doc as XmlDocument, root as XMLElement)*

This event allows you to add items to the XML data stream and they will become children of the root node. Use the LoadXMLItem event to restore the item when it is read back in.

## Working with the Code

There are two basic levels that you can work within the FTC. The first level is at the FormattedText class level. When you call methods on the FormattedText class, it will perform the action and then update the display. The second level is to call methods on the FTDocument class. The FTDocument methods will perform the action without updating the display. This is useful when you have a series of operations you need to do on the contents of the control and then call just one update at the end.

Shown below is the basic class hierarchy for the FTC (not all classes implemented in the FTC are shown). The classes highlighted in gray are classes you would implement.




## Creating Subclasses

*You should NOT modify the original source code, but instead create subclasses and make your changes to your subclasses. This will isolate your code changes from the base class changes in future releases of the FTC.*

If the FTC does not fulfill all of your requirements, you have a couple of options. The first is to use the provided events in the instance of the FTC to fill out the desired functionality. The second is to create subclasses of certain classes within the FTC. You should restrict yourself to subclassing the following classes--FormattedText, FTDocument, FTPage, and FTCustom.

To use subclasses of FTDocument or FTPage, you create the subclass and then instantiate the subclass in the NewDocument and NewPage events on the FormattedText instance or subclass respectively. The base code will call these events when it starts up to see if you created a subclass. If you don't return a subclass from these events, the base code will create the standard FTDocument and FTPage instances.

**Tag, you're it!**

Every object within the FTC hierarchy supports a developer defined tag that may be used for any purpose you see fit. The "tag" property is defined as a variant and you can use the tag property to store anything you want associated with the object. The FTC does no processing on the tag property.

In order to preserve your tag values across certain operations like cutting and pasting, you need to implement the FormattedText class events XmlEncodeTag and XmlDecodeTag to encode and decode your tags in the XML stream. See the demo application for an example on how to do this.

**WARNING:** Be careful to not create circular references with the tag property. For example, don't put direct object references to other objects within the FTC hierarchy. Great care has been spent assuring that the FTC class hierarchy is free from circular references to facilitate the release of the memory. If you need to refer to another object in the FTC hierarchy, use weak references or IDs and do lookups on the ID.

**Configuring the Software**

The FTC distribution is divided into two parts.

```
FTC Config
FTC Core
```

This division makes it easier to upgrade your projects when a new FTC release comes out. When you upgrade the FTC you can just upgrade the core files unless it is specifically stated in the release notes that the configuration files have been updated. The "FTC config" folder contains the files used to configure the FTC software. These configuration constants control the string, optimization, and debugging components of the FTC.

The first configuration file is called FTCPragma. The following constants are contained within this file.

```
FTC_BOUNDSCHECKING
FTC_NILOBJECTCHECKING
FTC_STACKOVERFLOWCHECKING
```

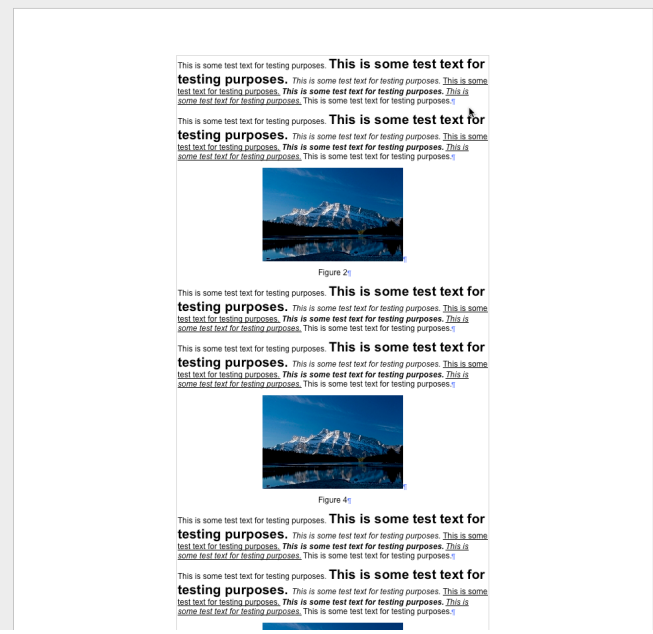
These constants control the #pragma directives used in every method in the FTC. By default these pragmas are turned off. You should not change them unless you are

running into a problem. If you do turn them on, you will be responsible for catching any exceptions that may be generated.

The FTCStrings module holds the constants for items that will be displayed such as undo/redo strings. You may customize these for different languages.

# Display Modes

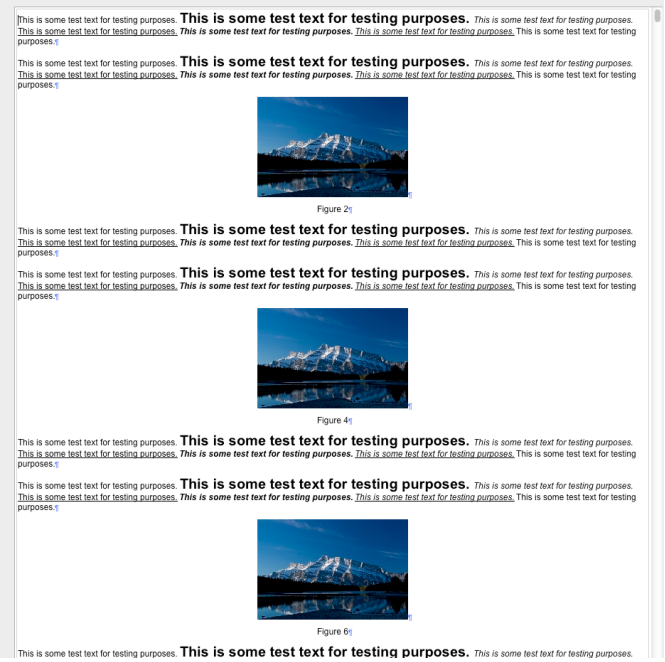
The FTC provides four modes of operation including page, normal, edit, and single line.



The page mode displays the text divided into pages. This is a What You See Is What You Get View and is very much like a Print Preview.

To change to Page View us the FormattedText.setPageViewMode command.

The normal mode is exactly like the page mode, but doesn't display page dividers or backgrounds. The normal mode is similar to Microsoft's Word normal mode.

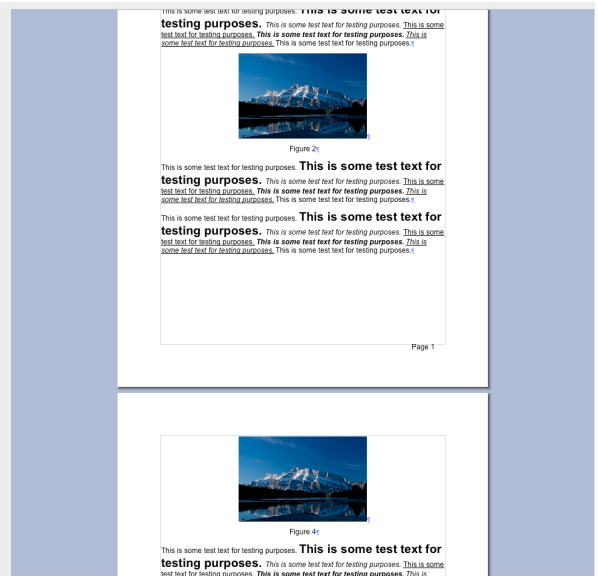


To change to Normal View use the FormattedText.setNormalViewMode command.

The edit mode binds the margins to the edges of the display. The edit mode is similar to the built in TextArea.  This is NOT a What You See Is What You Get view.

One thing to keep in mind when switching to Edit View (from a developers perspective) is that you will probably want to call the FormattedText.setEditViewMode method to reset the Edit Margins.  On the flip side, if you go back to Page or Normal view you will want to reset the

document margins to their real values.



To change to Edit View use the FormattedText.setEditViewMode command.

The single line mode is like the edit mode except there is no soft wrapping of lines. Each paragraph becomes a single line. Single line mode is useful if you are creating a code editor. You can set the mode at design time and you can switch between modes at run time.

In addition to the display modes, you can also change the alignment of the display for page and normal modes. It can be left, center, or right aligned.

## Replacing the TextArea

The FTC provides a connivence class called FTEditField. The FTEditField configures the FTC to behave like an TextArea. It sets the mode to edit and the margins to hug the edge of the display. At this point the FTEditField is like the TextArea except that it can display pictures and more! You can set the “AllowPictures” property to false to disallow pictures from being entered into the control and this causes it to more closely mimic the TextArea.

## Replacing the TextField

The FTC provides a replacement for the built in TextField called FTTextField. Its behavior is identical to the TextField except that it has all the advantages of the FTC like cross platform spell checking, undo capabilities and many more.

The LimitText property that limits the number of characters that can be typed into the control. The FTTextField also supports a mask for the control. Shown below are the symbols you can use in a mask. Note, the mask is case insensitive where an "a" is the equivalent of an "A" in the mask.

**#** - The single digit placeholder (0-9).

**A** - Alphabetic (a-z A-Z).

**B** - Alphanumeric (a-z A-Z 0-9).

**C** - Any printable character.

**[ ]** - Character set. Allows you to define a set of characters for that position. For example "[-+]" allows you to have a plus or minus sign for that particular character position.

**.** - Decimal placeholder. Derived from the international settings.

**,** - Thousands separator. Derived from the international settings.

**/** - Date separator. Derived from the international settings.

**** - Mask escape character. Used to specify literal characters that conflict with reserved mask characters.

**>** - Convert all the characters that follow to uppercase.

**<** - Convert all the characters that follow to lowercase.

All other symbols are displayed as literals for formatting purposes.

# XojoScriptField

If you implement XojoScript in your application you probably want a way for end users to edit XojoScript code within your application. With the FTXojoScriptField, you can support editing of XojoScript with syntax coloring, auto-indentation, line numbering, and error line indication. The FTXojoScriptField indents the code as you type and in the background as a thread to make sure all your code is properly indented.

With Syntax coloring, the following items can be set to individual colors.

- Comments
- Floating Point Numbers
- Integers
- Keywords
- Source Code
- Quoted Strings

You can also set the color of the error line indicator which is a color bar that spans the error line. Use the DefaultFont and DefaultFontSize properties to set the display font. You can also tell the FTXojoScriptField how to wrap the text to either no wrap or soft wrap.

Built into the FTXojoScriptField is a method to lookup error codes called lookupErrorCode. Use this method to translate compilation error codes into english equivalents for display to your end users.

# Embedding the FTC in a Canvas

You can embed multiple instances of the FTC within a Canvas control. This gives you editing capabilities within your custom control. To see it in action in the demo, invoke the menu item "File->New Test Window->Embedded FTC". The window that comes up will have two FTC controls within a single canvas. In the demo, you will want to start with the EmbeddedFTCWindow and look at the "Main" control.

Here are the basic steps for implementing the embedded FTC.

1. Create an invisible window to hold your instance of the FTC (see TestProxyWindow in the demo). Add an FTC instance and FTC scrollbars.

2. Create an FTCProxy instance. This acts as the go between the invisible window that contains the FTC that does the work and the canvas you want to display it on.

3. In the events of the canvas, call the proxy methods to pipe the events to the FTC. The methods you want to call start with the word "call". Here is the list of proxy events you need to call in your canvas events.

```
callConstructContexualMenu
callContextualMenuAction
callDropObject
callEnableMenuItems
callGotFocus
callKeyDown
callLostFocus
callMouseDown
callMouseDrag
callMouseEnter
callMouseExit
callMouseMove
callMouseUp
callMouseWheel
```

4. In your private invisible window, implement the events in the FTC to customize it to your needs (like adding custom object creation etc.).

That is basically all you need to do to start using it. Of course you will need to implement extra logic to handle the specifics of your application's needs with the embedded FTC like controlling the subfocus etc.

Here is a list of helper methods within the proxy you can use to complete your integration of the FTC into your custom canvas control.

```
isPointInControl(x as integer, y as integer) As boolean
```

Returns true if the coordinates are in the FTC.

```
resize()
```

Resizes the FTC position and dimensions. You should call this after you set the position or resize the dimensions.

```
setDimensions(width as integer, height as integer)
```

Set the width and height of the FTC.

```
setPosition(x as integer, y as integer)
```

Set the position of the FTC on the canvas.

```
update(compose as boolean)
```

Update the display of the FTC. If you pass true for the compose parameter, if forces the FTC to recompose the text. If you call it with compose set to false, it just blits the FTC image to the canvas.

You also have direct access to the FTC control through the “ftc” property in the proxy.

# FTC and Menus

The FTC supports Edit menu items including EditClear, EditCopy, EditCut, EditPaste, EditRedo, and EditUndo. The EditRedo menu item is not a standard menu item and you will need to add it to your project for the project to compile. You are not required to use the EditRedo menu item and you may hide it from the user.

# Supported Key Combinations

Shown below is a table of the supported key combination in the FTC. The option key under the Mac OS is mapped to the control key under Windows.

| Key | Shift | Option (control) | Action |
|---|---|---|---|
| up arrow | | | Move the insertion caret up one line in the same relative position. |
| | | x | Move to the beginning of the paragraph. |
| | x | | Extend the selection up one line. |
| | x | x | Select to the beginning of the paragraph. |
| down arrow | | | Move the insertion caret down one line in the same relative position. |
| | | x | Move to the end of the paragraph. |
| | x | | Extend the selection down one line. |
| | x | x | Select to the end of the paragraph. |
| left arrow | | | Move the insertion caret one character to the left. |
| | | x | Move to the beginning of the current word or next word. |
| | x | | Extend the selection left by one character. |
| | x | x | Select to the beginning of the current word or previous word. |
| right arrow | | | Move the insertion caret one character to the right. |
| | | x | Move to the end of the current word or next word. |
| | x | | Extend the selection right by one character. |
| | x | x | Select to the end of the current word or next word. |
| home | | | Move the insertion caret to the beginning of the line. |
| | | x | Move to the beginning of the document. |

| Key | Shift | Option (control) | Action |
|---|---|---|---|
| | x | | Select to the beginning of the line. |
| | x | x | Select to the beginning of the document. |
| end | | | Move the insertion caret to the end of the line. |
| | | x | Move to the end of the document. |
| | x | | Select to the end of the line. |
| | x | x | Select to the end of the document. |
| delete | | | Delete the previous character. |
| | | x | Delete the entire word preceding the insertion caret or if in the middle of a word, the preceding part of that word. |
| forward delete | | | Delete the next character. |
| | | x | Delete the entire word following the insertion caret or if in the middle of a word, the last part of that word. |
| page up | | | Scroll the display up one display full. |
| page down | | | Scroll the display down one display full. |

# Button and Toolbar Support

To implement buttons and toolbars to invoke FTC functionality, the following methods are available for querying the FTC for common attributes of the current selection or if there is no selection the attributes of the location of the insertion caret.

```
selectionAlignment
selectionBackgroundColor
selectionBold
selectionFontColor
selectionFontName
selectionFontSize
selectionItalic
selectionHyperlink
selectionHyperlinkColor
selectionLineSpacing
selectionMarked
selectionStrikethrough
selectionSubscript
selectionSuperscript
selectionUnderline
```

To query the FTC about the current availability of the edit menu items use the following methods.

```
isClearActionAvailable
isCopyActionAvailable
isCutActionAvailable
isPasteActionAvailable
isRedoActionAvailable
isUndoActionAvailable
```

To invoke the edit menu actions programmatically call the following methods.

```
editClearAction
editCopyAction
editCutAction
editPasteAction
editRedoAction
editUndoAction
```

To Change the case of the selected text, call the following methods.

```
changeToLowerCase
changeToUpperCase
changeToTitleCase
```

# Attributes

There are three levels where attributes of an FTC document are declared including the document, paragraph, and object levels. These attributes are stored in the FTDocument, FTParagraph, FTStyleRun, and FTObject classes respectively.

### Document Attributes

The document attributes control the overall look and feel of the displayed page.

Page dimensions:
- page width
- page height

Margins:
- left margin
- right margin
- top margin
- bottom margin

### Paragraph Attributes

The paragraph attributes control the size, relative position, and feel of the paragraph.

Margins:
- before space
- after space
- first indent
- left margin
- right margin

Miscellaneous:
- alignment
- background color
- line spacing
- page break
- paragraph style

### Character Attributes

The character attributes controls the look and feel of the displayed text.

Styles:
- bold
- font
- font size
- italic
- superscript
- subscript
- strikethrough
- underline

Colors:
- text color
- background color

Miscellaneous:
- character style
- default character style
- misspelled word indicator
- mark color

# Setting Defaults

The FTC has built-in default settings that you can override. The first line of settings you can override are the design time settings for the control. The next level of control is to implement a default paragraph style. To implement a default paragraph style you need to implement DefaultParagraphStyle event on your control or in your subclass. In this event you can set anything you want in terms of paragraph style settings. Shown below is some sample code you can put in the DefaultParagraphStyle event.

```
Dim ps as FTParagraphStyle

' Create a paragraph style.
ps = new FTParagraphStyle(me, "")

' Set some default attributes.
ps.setAlignment(FTParagraph.LEFT_ALIGNMENT)
ps.setBold(true)
ps.setAfterSpace(0.5)
' ...

' Return the paragraph style.
return ps
```

# Working with Styles

The FTC supports character and paragraph styles. Paragraph and character styles are stored in the FTDocument class. The FTC provides the infrastructure for you to use them, but does not implement how you should use them. In other words, the FTC provides the hooks, but not the glue code. How you implement the usage of style functionality is up to you based on the requirements of your target application.

Each FTParagraph contains a reference ID for the paragraph style so that you can go through and apply style changes when needed. You can use the setParagraphStyle and getParagraphStyle to store and retrieve the style ID for each paragraph. When a change is made to a paragraph style you will need to scan through the paragraphs and where the style ID that matches, you will then need to go in and change the appropriate style runs within the paragraph with the style information. The FTStyleRun class contains a reference ID for a character style and the same principles that apply to maintaing the paragraph style characteristics, also apply to character styles.

# Insertion Points

An insertion offset is made up of two parts including the paragraph index and the offset into the paragraph. Both the paragraph index and offset are zero based. The very beginning of the document would be represented as (0, 0). If you want to select characters 3 through 5 (one based), it would be represented as (0, 2) through (0, 5) with insertion points. Insertion point values fall in between characters.

An insertion point also has a “bias” that indicates where the insertion caret is to be displayed. The end of a line and the beginning of the next line have the same logical insertion offset. The bias distinguishes where to draw the caret and setting the bias to true means to use the beginning of the line.

## Adding Content

Sometimes you will want to add content to the FTC programmatically. Shown below is a simple example of how to add a paragraph and a line of bold text to the paragraph.

```
Dim sr as FTStyleRun
Dim p as FTParagraph

' Create the paragraph.
p = MyControl.getDoc.addNewParagraph

' Create a style run.
sr = new FTStyleRun(MyControl)

' Set the characteristics for the style run
sr.font = "geneva"
sr.fontSize = 12
sr.bold = true

' Add the text to the style run.
sr.setText("This is some test text for testing purposes.")

' Add the style run to the paragraph.
p.addObject(sr)

' ...rinse, lather, and repeat the above code to build up the content...

' Update the display.
MyControl.update(true, true)
```

See the ‘Create Document’ demo application for a more detailed working example. You can also add pictures, custom items, and tabs. Here are some sample code lines you would substitute in the above code.

```
p.addObject(new FTPicture(parentControl, MountainSmall))
p.addObject(new FTTab(parentControl), false)
```

When you add an object, it appends it to the end of the paragraph. You can also insert objects anywhere like in a paragraph by using the equivalent insert methods.

Within the FormattedText class there are several methods used for updating the display.

*update*

This method schedules an update to occur the next time an update is required to update the insertion caret. There can be a slight visual delay.

*update(compose as boolean, force as boolean, selectionOnly as integer = 0)*

This is the main method you should call to update the display. The compose parameter forces the composition of paragraphs that have been marked dirty. The force parameter controls drawing of the content. The selectionOnly parameter is used only in controlling the updating selections and in general should not be used.

*updateFull*

This method updates the entire control. This an expensive method to call because it recomposes the entire document. So be careful when you call it.

# Setting Tabs

The FTC supports left bound tab stops only. Tabs are internally divided into two parts. The first part is the tab stop which acts like a framework where tabs are placed and the second part are the actual tabs which the user types.

Tab stops are generated from two different sources. The first is the default tab stop specified in the FormattedText class as specified with the DefaultTabStop property. The second source of tab stops is paragraph specific tab stops. To add a tab stop to a paragraph, call the FTParagraph.addTabStop method.

# Changing Styles

The FormattedText class provides a series of “change” methods that allow you to change a text characteristic of the currently selected text, the word the insertion caret resides in, or setup the style for the next character to be typed. These methods can be called at the FormattedText level or the FTDocument level. Calling them at the FormattedText level always invokes an update.

Here is the list of change methods.

```
changeBackgroundColor
changeBoldStyle
changeCenterAlignment
changeCharacterStyle
changeFont
changeHyperlink
changeHyperlinkColor
```

```
changeItalicStyle
changeLeftAlignment
changeLineSpacing
changeMarginGuides
changeMark
changeOpacity
changeParagraphBackgroundColor
changeParagraphMarigns
changePlainStyle
changeRightAlignment
changeShadwo
changeSize
changeStrikeThrough
changeSubscript
changeSuperscript
changeTextColor
changeUnderlineStyle
```

# Accessing the Text

The FTDocument class contains the data that is displayed by the control. You can access this data through several different methods. The text is stored as a list of FTParagraphs and you can use the getParagraph method to access them. The data contained within the FTParagraph object is the un-composed form (not parsed into lines). You can use this form to do things like search for words and change the content. If you change the content, make sure you call one of the update methods to recompose the text in the display. If you need to access the data on a line by line basis (as seen in the display), each paragraph contains a list of FTLine objects after the paragraph has been composed.

### Manually

Shown below is a simple search function that accesses the FTDocument data.

```
Function findWord(doc as FTDocument, startParagraph as integer, startOffset
as integer, word as string) As boolean

  #pragma DisableBackgroundTasks

  Dim i as integer
  Dim count as integer
  Dim position as integer
  Dim text as string
  Dim p as FTParagraph

  ' Get the number of paragraphs.
  count = doc.getParagraphCount - 1

  ' Scan the paragraphs.
  for i = startParagraph to count

    ' Get a paragraph.
    p = doc.getParagraph(i)
```

```
    ' Get the text from the paragraph.
    text = p.getText(true)

    ' Search for the word.
    position = inStr(startOffset, text, word)

    ' Use the start of the paragraph.
    startOffset = 0

    ' Did we find it?
    if position > 0 then

      ' Make it zero based.
      position = position - 1

      ' Select the word.
      doc.selectSegment(i, position, i, position + word.len)

      ' Update the display.
      doc.parentControl.update(true, true)

      ' We are done.
      return true

    end if

  next

  ' We did not find it.
  return false

End Function
```

### Automatically

The same task of scanning the FTC content can be accomplished with the FTIterator and FTIteratorBase classes. These classes handle walking through the  FTC data structures and allow you to concentrate on handling the paragraphs and style runs. Shown below is an example that creates HTML output from the FTC data structures (see the menu item Export->HTML in the demo application).

```
' Create the file.
tos = TextOutputStream.Create(fi)

' Create the HTML output.
it = new HTMLIterator
ftit = new FTIterator(TestDisplay, it)
ftit.run(nil, nil)

' Write out the HTML.
tos.Write(it.getHTML)
```

The HTMLIterator class (see the HTML Converter demo application) is a subclass of the FTIteratorBase class. The FTIterator class uses classes based off FTIteratorBase to provide a set of events you can fill in to handle the data as it encountered. You can call

the the run method specifying the range of text to be processed. For example, ftit.run(nil, nil) means use the whole content of the document.

Note, the data accessed by the FTIteratorBase is a copy of the original. This means modifying the style runs within your subclass will not affect the original data structures in the FTC.

## Search and Replace

With the FTC you can search and replace interactively or programatically. See the "SearchWindow" window in the demo for an example of how to implement interactive search and replace it in your application.

The search method allows you to specify the starting and ending locations for the search. If you pass in nil for searchStart or searchEnd then the beginning or end of the document will be used respectively.

```
search(target as string, searchStart as FTInsertionOffset, searchEnd as
FTInsertionOffset, ByRef selectStart as FTInsertionOffset, ByRef selectEnd as
FTInsertionOffset) as boolean
```

The scrollToSelection method allows you to make sure the current selection is visible in the display.

```
scrollToSelection
```

To programmatically replace all occurrences of a string with another string, use the following method.

```
replaceAll(target as string, replacement as string, searchStart as
FTInsertionOffset, searchEnd as FTInsertionOffset)
```

The same behavior applies to the searchStart and searchEnd in the replaceAll method as does in the search method.

## Spell Checking

There are three basic ways to perform spell checking which includes through a dialog which you supply (modal spell checking), spell check as you type, or as a background thread. What type of spell checking methods you choose depends on the context of your application.

If you implement modal spell checking, you can get the target text of the paragraph by calling FTDocument.getParagraph to get the paragraph and then calling FTParagraph.getText to get the actual text. To mark a word as misspelled, call the FTParagraph.changeMisspelledWord method and then update the display.

The FTC provides a mechanism for spell checking as a background thread. The FTC creates a thread when it starts up which will wait until it is invoked via the FormattedText.checkSpelling method. This thread will then start at the beginning of the document and spell check all of the paragraphs and will call the SpellCheckParagraph event for each paragraph. Inside the SpellCheckParagraph is where you do the actual spell checking. Every time the checkSpelling method is called, it will restart from the beginning of the document.

To perform spell checking as you type, you need to turn it on with a property called SpellCheckWhileTyping. You also must fill in the SpellCheckWord and SpellCheckParagraph events do the actual spell checking.

The second half of the spell check equation is doing the actual spell checking. Spell checking is a very difficult discipline to program and creating your own spell checker from scratch is a fools errand unless you are going into the spell checking business. If you only want to use the FTC on Mac OS X, then the Einhugur and MBS plugins provide relatively cheap spell checking solutions. These solutions tap into the built in spell checker in the OS and are adequate for most applications, but lack advanced features.

If you require an advanced, cross-platform solution, then BKeeney Software provides a plugin based on the Hunspell spell checking library. The package including a plugin can be downloaded from the following URL.

https://www.bkeeney.com/allproducts/bks-spell-checker-plugin-for-xojo/

The Spell Checking plugin may optionally use Hunspell dictionaries (can be found in various locations throughout the internet) for Mac OS or Windows. However, in all versions of MacOS and in Windows 8 and better the plugin can be configured to use the built-in system dictionaries. It is possible to configure your app to use Hunspell on those systems that don't have a built-in spell checker (Windows 7) and the system spell checker at other times. Linux is not supported.

Without a license, the spell check plugin is fully functional in the debugger but will not check words in a built application. You can purchase the plugin from BKeeney Software for $100. The Spell Checking demo application uses this spell checker and requires that the plugin be installed in your Xojo plugins folder.

# Hyperlinks

Hyperlinks can be added to a document by calling the ChangeHyperlink method and by passing in the URL of the hyperlink. To reset the hyperlink just pass in an empty string. It is up to the developer to actually DO something with the hyperlink if the user clicks on it. You can handle it via the HyperlinkClicked event.

# Printing

Printing is done by calling the print method in the FormattedText class. You must pass in the print graphics object for the printer and the PrinterSetup object. See the print menu item in the demo application for an example of how to print from your application.

# Saving the Data

### Formats

The FTC supports saving and opening the document data in three different formats including plain text, RTF, and the native FTC XML format. Each format has its advantages and disadvantages and you can choose to support any of the formats in your end application.

When you save the data in plain text or RTF, you will loose formatting information. With plain text you will loose all formatting information and any images that were in the document. With RTF you will loose FTC specific features such as custom items based on the FTCustom class (custom items will be converted to pictures).

The FTC's responsibility is to produce and parse the data, but it is your application's responsibility to retrieve and store the data. When you get the XML data form the FTC, it is recommended that you compress it before saving it to a file to save space. Whatever encoding you apply to the data you should decode before sending it to the FTC.

There are six utility methods you can use to convert the plain text, RTF and XML data into the other forms of data. You will find them in the FTUtilities module.

- convertRTFToText
- convertRTFToXML
- convertTextToRTF
- convertTextToXML
- convertXMLToRTF
- convertXMLToText

### Plain Text

When you save the document contents as plain text, pictures or custom items will be converted to spaces.

### RTF

Within the FTC distribution is an RTF engine. This engine is independent of the FTC itself and can be used separately. The FTC provides glue code between this engine and the FTC and you would need to provide similar glue code if you use it by itself.

The RTF engine is designed to serve the FTC so it supports those features in the RTF specification. All other RTF features will be ignored.

Note, that the RTF engine is invoked when retrieving or sending content to the clipboard or when dragging and dropping.

### XML

The native XML format is a lossless format in that there will be a complete restoration of the document's data when it is read in. The XML format is a completely self contained format and no external references or documents are required.

The general format of the XML format is as follows.

```
<root>

      <version>

      <FTDocument>

      <ParagraphStyles>
            <FTParagraphStyle>...

      <CharacterStyles>
            <FTCharacterStyle>...

      <Paragraphs>
            <FTParagraph>...
                  <FTStyleRun>
                  <FTPicture>
                  <FTTab>
                  <User defined nodes - FTCustom>...

      <your nodes>...
```

Most of the data stored in the format is stored as attributes on the XML nodes. For example, here is a typical style run which stores a piece of text with its associated style information.

```
<FTStyleRun bc="255 255 255" bcd="false" f="arial" s="12" os="12" tc="0 0 0"
cs="0" t="This is some test text for testing purposes."/>
```

The attributes are used like name value pairs. In this example only a portion of the possible attributes are shown because attributes are assumed to be set to default values within the FTC. Broken out into a more readable form they would be...

```
Background color = white
Background color = not defined
Font = arial
Font size = 12
Original size (for sub/super scripting) = 12
Text color = black
Character Style = 0
Text = This is some test text for testing purposes.
```

If you need to examine the XML data produced by the FTC, the program “XML Editor” from “www.elfdata.com” is an excellent tool.

### XML Version

At the beginning of the XML stream is a node called “version”. This is where your application specific identification data is stored. There are four fields that can be set including creator, document tag, file type, and file version. To set these fields use the setXMLCreator setXMLDocumentTag setXMLFileType and setXMLFileVersion methods. You can use the corresponding get methods to retrieve the values after the document has been loaded. You may use these field in any manner you wish (the FTC does not do anything with these fields).

## XML Attributes

The FTCxml module contains all the constant definitions for the XML names and attributes. Shown below is a break down of the attributes for each XML element.

### Version

Attributes:
- Creator = "c"
- Document Tag = "dt"
- File Type = "ft"
- File Version = "fv"

### FTDocument

Attributes:
- Bottom Margin = "bm"
- Left Margin = "lm"
- Page Background Color = "pbc"
- Page Height = "ph"
- Page Width = "pw"
- Right Margin = "rm"
- Top Margin = "tm"

### ParagraphStyles

Attributes:
- After Space = "as"
- Alignment = "a"
- Background Color = "bc"
- Background Color Defined = "bcd"
- Before Space = "bs"
- Bold = "b"
- First Indent = "fi"
- First Indent Defined = "fid"
- Font = "f"
- Italic = "i"
- Left Margin = "lm"
- Left Margin Defined = "lmd"
- Line Spacing = "ls"
- Mark Color = "mc"
- Mark Defined = "md"
- Mark On = "mo"
- Right Margin = "rm"
- Right Margin Defined = "rmd"
- Script Level = "sl"
- Script Level Defined = "sld"
- Size = "s"
- Strikethrough = "st"
- Style ID = "sid"
- Style Name = "n"
- Text Color = "tc"
- Text Color Defined = "tcd"
- Underline = "u"

### CharacterStyles

Attributes:
- Background Color = "bc"
- Background Color Defined = "bcd"
- Bold = "b"
- Font = "f"
- Italic = "i"
- Mark Color = "mc"
- Mark Defined = "md"
- Mark On = "mo"
- Script Level = "sl"
- Script Level Defined = "sld"
- Size = "s"
- Strikethrough = "st"
- Style ID = "sid"
- Style Name = "n"
- Text Color = "tc"
- Text Color Defined = "tcd"
- Underline = "u"

**FTParagraph**

Attributes:
- After Space = "as"
- Alignment = "a"
- Background Color = "bc"
- Before Space = "bs"
- First Indent = "fi"
- Left Margin = "lm"
- Line Spacing = "ls"
- Page Break = "pb"
- Paragraph Style = "ps"
- Right Margin = "rm"
- Tab Stop = “ts”

**FTPicture**

Attributes:
- Background Color = "bc"
- Background Color Defined = "bcd"
- Data = "d"
- Height = "h"
- Mark Color = "mc"
- Marked = "m"
- Scaled Height = "sh"
- Scaled Width = "sw"
- Strikethrough = "st"
- Text Color = "tc"
- Width = "w"

**FTStyleRun**

Attributes:
- Background Color = "bc"
- Background Color Defined = "bcd"
- Bold = "b"
- Character Style = "cs"
- Font = "f"
- Hyperlink = “cushl”
- Hyperlink Color = “cushlc”
- Hyperlink Color Disabled = “cushlcd”
- Hyperlink Color Rollover = “cushlcr”
- Hyperlink Color Visited = “cushlcv”
- Italic = "i"
- Mark Color = "mc"
- Marked = "m"
- Original Size = "os"
- Shadow = “cuhs”
- Shadow Angle = “cuha”
- Shadow Color = “cuhc”
- Shadow Offset= “cuho”
- Shadow Opacity = “cuhop”
- Size = "s"
- Strikethrough = "st"
- Subscript = "sbs"
- Superscript = "sps"
- Text Encoded = “te”
- Text = "t"
- Text Color = "tc"

**FTTab**

Attributes:
- Background Color = "bc"
- Background Color Defined = "bcd"
- Mark Color = "mc"
- Marked = "m"
- Strikethrough = "st"
- Tab Stop = "ts"
- Tab Type = "tt"
- Text Color = "tc"

# Custom Objects

The FTC supports the creation of custom made objects to display/edit application specific data. Custom objects are a subclass of the FTCustom class which inherits from the FTPicture class and thus inherits the functionality of pictures. To create a custom object you create a subclass of the FTCustom class.

When you need to update the display of your custom object from within your custom object, call the updateContents method instead of the control's update methods. Another method to be aware of is the FTPicture.isResizable method. Override this method want to control if the control should be resizable (display grab handles).

### Saving the Data

Each custom object will have specific data that needs to be saved and restored. To accomplish this you need to implement the OpenXML and SaveXML events within the object. This is where you encode your data and attach it to the node in the XML stream.

There are two tasks that are up to implement. The first is to create the object by creating a new instance of the object and passing whatever data you need into it. Then you have to find the Paragraph you want to add it to.

The best way to save your document to keep these custom objects is XML. When they are read back into the control via the SetXML method, the control will fire the XMLObjectCreate event where you pass the XML Element Data into your constructor that can handle and decode that data.

When you save your document to the RTF format, the custom object will be converted to a picture and you will lose the specific custom data. If you reopen the saved RTF document, the former custom object will now be a plain picture. So you don't need to do anything in the SaveRTF event, but it is there if you want do something special.

For an example, please see the CustomObject Demo project. In this example, dragging a file onto the control will grab the file, copy it to your Application Data/FTC Custom Object/ directory, get a copy of its icon (assuming you have the Einhugur IconLib plugin installed) and then displays the Custom Object in the control. The user can then right-click (contextual click) on the object for a Contextual Menu where the user can find on disk or launch.

### Events

The following events are available for you to drive your custom object. These events are the similar in meaning as you would find in any other Xojo control.

*DropObject(obj as DragItem, action as integer)*

An object was dropped on the control.

*GotFocus() as integer*

The control has gained the focus. To receive idle events, you need to return a positive value which represents the time interval between idle events in milliseconds.

*Idle() as integer*

This event allows you to perform periodic processing. You can change the period of the events by returning positive value which represents the time interval between idle events in milliseconds. If you return zero, the previous setting remains in place.

*KeyDown(key as string) as boolean*

A key was pressed.

*LostFocus()*

The control has lost the focus.

*MouseDown(x as integer, y as integer)*

The mouse button was pressed. The FTCustom object tracks the double click time and will call the FormattedText.ObjectDoubleClick event if the object detects that it has been double clicked.

*MouseDrag(x as integer, y as integer) as boolean*

The mouse was clicked and subsequently dragged.

*MouseEnter()*

The mouse has moved within the confines of the controls.

*MouseExit()*

The mouse has moved outside the confines of the control.

*MouseMove(x as integer, y as integer)*

The mouse position has changed.

*MouseUp(x as integer, y as integer)*

The mouse button was let up.

*MouseWheel(x as integer, y as integer, deltaX as integer, deltaY as integer) as boolean*

The user moved the mouse wheel.

*Open()*

The super.Constructor has been called and this is your chance to do any further initializations.

*OpenXML(node as XmlElement)*

The object is being loaded from an XML data stream and this event is where you parse the XML data to restore the content of the custom object. This is called before the super.Constructor.

*Paint(g as Graphics, print as boolean, printScale as double)*

This event is called when the FTC requires control to refresh itself. It is suggested that you double buffer the display of the control. You can accomplish this by either using the built-in facilities within the FTPicture class to store the display picture or implement your own double buffering. Draw the actual content when the content changes or is resized.

*Resize(g as Graphics)*

The control is being resized.

*Resized(g as Graphics)*

The control has finished being resized.

*SaveRTF(rtf as RTFWriter) as boolean*

This event is where you save the contents of your custom object to the RTF data stream. As a custom object, you will lose data in that your custom object will be converted to a picture (if you do nothing).

*SaveXML(xmlDoc as XmlDocument) as XmlElement*

This event is where you save the contents of your custom object to the XML data stream. How you encode and store the data on the node is your preference.

# Frequently Asked Questions

- **How can I incorporate carriage returns (Chr(13)) into the an FTC document?**

  In the FTC carriage returns (CR) are represented by paragraphs (FTParagraph). So within the text of a paragraph you would not have embed CRs (it would not make sense). So a string that contains CRs needs to be parsed and split into paragraphs. There are several ways to do this. You could use the setText or insertString methods. They will automatically parse the CRs and create the paragraphs for you. Also see “Adding Content” in this document.

- **Is there way to determine the word that was selected in the right click and then automagically select it?**

  If you want to determine if the cursor was within the selected word when the user right clicks, you can use findSelectionRectangle to get the selection rectangle and then compare the X, Y coordinates and see if it is within the word. If it then then falls within the rectangle, then you can get the selected text and do any further processing as needed.

  Now when there is no selected text and the user clicks on a word, you need to call FTDocument.findWordBoundaryAtInsertion. This looks at the current insertion point and gives you the boundary of the word the user just clicked on.

- **The FTC is not responding to typing.**

  It’s a common problem when copying Xojo objects for some properties not to get copied.  Make sure you have the AcceptFocus property set to true.

- **How do I implement Select All?**

  To connect your FormattedTextControl to the Edit->SelectAll Menu is relatively easy.  Add a menu handler for your EditSelectAll menu command in the window where your control resides.  Then, if your control is named ftc1 you’d have something like this:

```
Function EditSelectAll() As Boolean
  ftc1.selectAll
  return true
End Function
```

- **Getting the Height of the Document**

  To determine the height of the user content use the FTDocument.GetTotalPageLength function.  It returns the total height of the document.

So to get the height of yrou content you'd need to do something like this (where ftc is your FormattedText control):

```
dim dTotalHeight as double
dTotalHeight = ftc.getDoc.getTotalPageLength
ftc.Height = dTotalHeight
```

You'll probably want to include a sensible maximum size. Since there's no fixed limit to the document size you could end up with a very large control if a user does something unexpected like pasting in the entire contents of a book or something.

- **Changing the page color**

  If you want to change the page color of the FTC you can set it using the FTDocument.SetBackgroundColor method.

  Example:  In the open event of your Formatted Text Control Instance you can use this:

```
me.GetDoc.setPageBackgroundColor &cFFFF00
```

# Legal

## Copyright Notices



## End User License Agreement

CAREFULLY READ THE FOLLOWING LICENSE AGREEMENT. YOU ACCEPT AND AGREE TO BE BOUND BY THIS LICENSE AGREEMENT BY CLICKING THE ICON LABELED "I ACCEPT" THAT IS DISPLAYED WHEN PURCHASED. IF YOU DO NOT AGREE TO THIS LICENSE, CLICK THE ICON LABELED "CANCEL" AND YOUR ORDER WILL BE CANCELED, THE SOFTWARE WILL NOT BE DOWNLOADED AND YOU WILL NOT BE CHARGED.

## License Grant

"You" means the person or company who is being licensed to use the Software or Documentation. "We," "us" and "our" means "BKeeney Software Inc."

We hereby grant you a nonexclusive license to use one copy of the Software on any single computer, provided the Software is in use on only one computer at any time. The Software is "in use" on a computer when it is loaded into temporary memory (RAM) or installed into the permanent memory of a computer—for example, a hard disk, CD-ROM or other storage device.

If the Software is permanently installed on the hard disk or other storage device of a computer (other than a network server) and one person uses that computer more than 80% of the time, then that person may also use the Software on a portable or home computer.

## Title

We remain the owner of all right, title and interest in the Software and related explanatory written materials ("Documentation").

## Archival or Backup Copies

You may copy the Software for back up and archival purposes, provided that the original and each copy is kept in your possession and that your installation and use of the Software does not exceed that allowed in the "License Grant" section above.

### Things You May Not Do

The Software and Documentation are protected by United States copyright laws and international treaties. You must treat the Software and Documentation like any other copyrighted material—for example, a book. You may not:

- copy the Documentation,
- copy the Software except to make archival or backup copies as provided above,
- reverse engineer, disassemble, decompile or make any attempt to discover the source code of the Software,
- place the Software onto a server so that it is accessible via a public network such as the Internet,
- sublicense, rent, lease or lend any portion of the Software or Documentation, or
- distribute the Software or any of its parts as a part of a component, a library or a developer's toolkit.

### Transfers

You may transfer all your rights to use the Software and Documentation to another person or legal entity provided you transfer this Agreement, the Software and Documentation, including all copies, updates and prior versions to such person or entity and that you retain no copies, including copies stored on computer.

### Custom Software Development

If you develop software for any other party (for example "custom software development") with the exclusion of use by Your End User, then you are required to

1) confirm that the other party also has a legal license from BKeeney Software Inc. for the same Product and that the serialization information to initialize the Product is that of the third party, and not yours;
2) notify the other party that they must agree to the then current version of this Agreement before any transfer of the Work.

For example, if you create an application for XYZ Corporation and you provide the source code for the application which includes this Software, then you must confirm that XYZ Corporation also has a legal license to the Software from BKeeney Software Inc. before you may transfer the Software to XYZ Corporation.

### No Inconsistent Right

You may not grant to Your End User any rights that are inconsistent to the rights granted to you in this Agreement. You may not grant to Your End User the right to distribute any portion of the Distributable Components separately from your Work, or provide them with any assistance, consulting or information that will allow them to do so. You agree to not attempt to modify or change the initialization procedure or assist, consult, direct or inform any other entity on methods to modify or change the initialization procedure.

### Limited Warranty

We warrant that for a period of 90 days after delivery of this copy of the Software to you:

the media on which this copy of the Software is provided to you will be free from defects in materials and workmanship under normal use, and the Software will perform in substantial accordance with the Documentation.

To the extent permitted by applicable law, THE FOREGOING LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES OR CONDITIONS, EXPRESS OR IMPLIED, AND WE DISCLAIM ANY AND ALL IMPLIED WARRANTIES OR CONDITIONS, INCLUDING ANY IMPLIED WARRANTY OF TITLE, NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, regardless of whether we know or had reason to know of your particular needs. No employee, agent, dealer or distributor of ours is authorized to modify this limited warranty, nor to make any additional warranties.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

**Limited Remedy**

Our entire liability and your exclusive remedy for breach of the foregoing warranty shall be, at our option, to either: return the price you paid, or repair or replace the Software or media that does not meet the foregoing warranty if it is returned to us with a copy of your receipt.

IN NO EVENT WILL WE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OR THE INABILITY TO USE THE SOFTWARE (EVEN IF WE OR AN AUTHORIZED DEALER OR DISTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF THESE DAMAGES), OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

**Term and Termination**

This license agreement takes effect upon your use of the software and remains effective until terminated. You may terminate it at any time by destroying all copies of the Software and Documentation in your possession. It will also automatically terminate if you fail to comply with any term or condition of this license agreement. You agree on termination of this license to destroy all copies of the Software and Documentation in your possession.

**Confidentiality**

The Software contains trade secrets and proprietary know-how that belong to us and it is being made available to you in strict confidence. ANY USE OR DISCLOSURE OF THE SOFTWARE, OR OF ITS ALGORITHMS, PROTOCOLS OR INTERFACES, OTHER THAN IN STRICT ACCORDANCE WITH THIS LICENSE AGREEMENT, MAY BE ACTIONABLE AS A VIOLATION OF OUR TRADE SECRET RIGHTS.

**General Provisions**

1) This written license agreement is the exclusive agreement between you and us concerning the Software and Documentation and supersedes any prior purchase order, communication, advertising or representation concerning the Software.
2) This license agreement may be modified only by a writing signed by you and us.
3) In the event of litigation between you and us concerning the Software or Documentation, the prevailing party in the litigation will be entitled to recover attorney fees and expenses from the other party.
4) This license agreement is governed by the laws of the State of Kansas.
5) You agree that the Software will not be shipped, transferred or exported into any country or used in any manner prohibited by the United States Export Administration Act or any other export laws, restrictions or regulations.