This is taken from https://www.codeproject.com/Articles/30936/Using-ICSharpCode-TextEditor.

My edits/adds are marked.

A text editor actually contains three nested controls that are closely coupled to one another:

At the top level is **TextEditorControl**, which contains either one or two **TextAreaControls**. It has two **TextEditorControls** when "split", as demonstrated in the screenshot.

**TextAreaControl** encapsulates the horizontal and vertical scroll bars, and a **TextArea**.

**TextArea** is the control that actually gets the focus. It paints the text and handles keyboard input.

If there's one thing more important than the control classes, it's the **IDocument** interface. The **IDocument** interface, implemented in the **DefaultDocument** class, is the hub that provides access to most of the features of SharpDevelop's text editor: undo/redo, markers, bookmarks, code folding, auto-indenting, syntax highlighting, settings, and last but not least, management of the text buffer.

The document provides unlimited undo/redo automatically. You need not do anything special to ensure that programmatic changes can be undone; just be sure to modify the document using methods in IDocument, not in ITextBufferStrategy (the latter bypasses the undo stack). You can group multiple actions together so that one "undo" command undoes them all by surrounding the group with matching calls to **IDocument**.UndoStack.StartUndoGroup() and **IDocument**.UndoStack.EndUndoGroup().

Markers (instances of the **TextMarker** class) are ranges of text (with a start and end position). After registering a marker with a document's **MarkerStrategy**, the marker's start and endpoints move automatically as the document is modified. Markers can be visible or invisible; if visible, a marker can either underline text (with a spellchecker-style squiggle), or override the syntax highlighting of the region it covers. The sample application uses markers to implement its "Highlight all" command.

*Curiously, there is another class which serves a similar purpose: TextAnchor anchors to a single point, and automatically moves as the document is changed, but you can't use this class because its constructor is internal.*

Bookmarks are rectangular markers shown in the "icon bar" margin, which the user can jump to by pressing F2. The sample project shows how to toggle bookmarks and move between them.

*The DefaultDocument and TextEditorControl do not try to update code folding markers automatically, so in the demo, folding is only computed when a file is first loaded.*

In the presence of code folding, there are two kinds of line numbers.

* "logical" line numbers which are the 'real' line numbers displayed in the margin.
* "visible" line numbers which are the line numbers after folding is applied. The term "line number" by itself normally refers to a logical line number.

Auto-indenting, and related features that format the document in reaction to the user's typing, are intended to be provided in an implementation of **IFormattingStrategy**. The **DefaultFormattingStrategy** simply matches the indentation of the previous line when Enter is pressed. Again, fancier strategies can be found in SharpDevelop's source code.

**IFormattingStrategy** also contains methods to search backward or forward in the document for matching brackets so they can be highlighted, but this is just part of the mechanism for highlighting matching brackets, a mechanism whose implementation spans several classes including TextUtilities, BracketHighlightingSheme, BracketHighlight, and TextArea*. Anyway, it appears that TextArea is hard-coded to provide brace matching for (), [], and {} only.*

The text buffer strategy manages the text buffer. The algorithm behind the default **~~GapTextBufferStrategy~~** is described on Wikipedia and on CodeProject.

The text editor library is very large; there are a number of other miscellaneous classes that couldn't fit on the diagram, which I don't have time to describe in this article. Notable ones include **TextWord**, the atomic unit of syntax highlighting; **LineManager**, which DefaultDocument uses to convert "offsets" to "positions"; and **TextUtilities**, a collection of static methods.

Here are some more tips:

A location in a document can be represented in two ways. First, a location can be represented as a line-column pair, which one bundles together in a TextLocation structure. More fundamentally, you can think of a document as an array of characters whose length is **IDocument**.TextLength. An index into this array is called an "offset" (type: int). The offset representation seems to be more common, but some code (e.g., the **SelectionManager**) requires locations to be supplied in the form of TextLocations. You can use IDocument.OffsetToPosition and IDocument.PositionToOffset to convert between the two representations.

The "**Caret**" is the flashing cursor. You can move the cursor by changing the Caret's Line, Column, or Position properties.

All text editor actions that can be invoked with a key combination in SharpDevelop are encapsulated in implementations of ICSharpCode.TextEditor.Actions.**IEditAction**. A few of these actions are demonstrated in the example application's Edit menu handlers.

The left side of the **TextArea** shows up to three margins, represented by three classes that are not on the diagram above. They are not separate controls, but **TextArea** passes mouse and paint commands to them.

* **FoldMargin** shows the little + and - icons for collapsing or expanding regions. If you don't use code folding, I'm afraid there is no way to hide the margin (well, you could change the source code).
* **IconBarMargin** shows icons such as bookmarks (or breakpoints in SharpDevelop). Visibility is controlled by ITextEditorProperties.IsIconBarVisible.
* **GutterMargin** shows line numbers. Visibility is controlled by ITextEditorProperties.ShowLineNumbers.

*The document has no reference to the controls that use it, so I assume we could use the same document in multiple controls, manage a document that has no control, or write a new control implementation. Editor controls are informed of changes to the document by subscribing to its events.*

*The most heavyweight part of ICSharpCode.TextEditor is its syntax highlighting, which can use ten times as much memory as the size of the text file being edited. Code to draw this text uses a lot of CPU power and allocates copious amounts of temporary objects.*