



SlickEdit® Gadgets

As a programmer, one of the best feelings in the world is writing something that makes you want to call your programming buddies over and say, "This is cool! Check this out." Sometimes it happens while developing a prototype or proof of concept. Sometimes it's just a crazy idea you decided to experiment with.

We have these moments at SlickEdit too, and decided to gather some of them up and put them out there for others to use. We call these SlickEdit Gadgets. Some of the gadgets provide useful utilities and some of them are just for fun. We hope that as you go through these, at least one of them will make you want to call your office mates over to check it out.

Feel free to stop by the SlickEdit Tools forum as well, to discuss the gadgets, and find out new and interesting ways to use them: <http://community.slickedit.com/>.

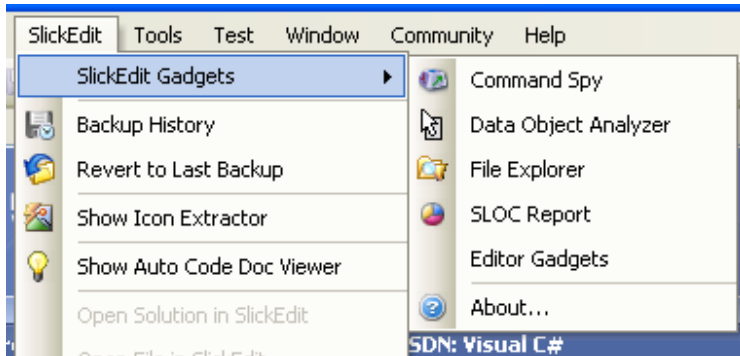
If you're interested in additional utilities like SlickEdit Gadgets, check out the SlickEdit Tools for Microsoft® Visual Studio® product: <http://www.slickedit.com/tools/>.

Contents

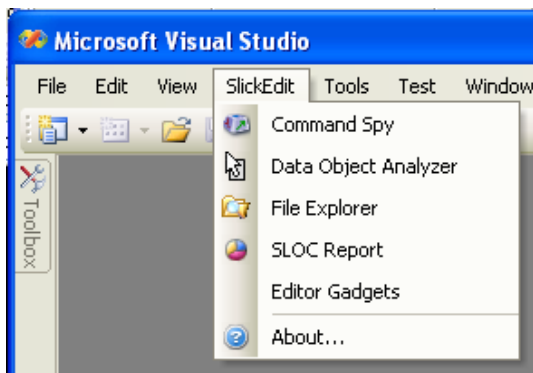
Installation Note	3
The Command Spy.....	4
<i>How does it work?.....</i>	4
<i>The toolbar.....</i>	5
The Data Object Analyzer.....	6
<i>How does it work?.....</i>	6
<i>Auto-Generating Data Format Handlers.....</i>	7
<i>The toolbar.....</i>	7
File Explorer	8
<i>How does it work?.....</i>	8
<i>Saving favorite folders</i>	8
<i>The toolbar.....</i>	9
Editor Gadgets	10
<i>How does it work?.....</i>	10
Line ruler.....	10
Indentation guide	11
Auto-copy selection.....	11
Editor graphics.....	11
<i>Configuration.....</i>	13
Line ruler.....	13
Indentation guide	13
Auto-copy selection.....	14
Editor Graphic.....	14
Performance	14
Graphic Recommendations.....	15
The SLOC Report	16
<i>How does it work?.....</i>	16

Installation Note

If you already have SlickEdit Tools installed, then a **SlickEdit Gadgets** sub-menu will appear in the **SlickEdit** menu, as shown in the following screen capture:



If you don't have SlickEdit Tools installed, then the individual gadgets will appear beneath a top level **SlickEdit** menu, as shown in the following screen capture:



All menu references for the gadgets will be shown as **SlickEdit > [name]** where [name] is the gadget's menu text.

The Command Spy

Whenever you click on a menu item or toolbar button in Visual Studio®, you are executing what is known as a “command”. Commands may be executed in one of three ways:

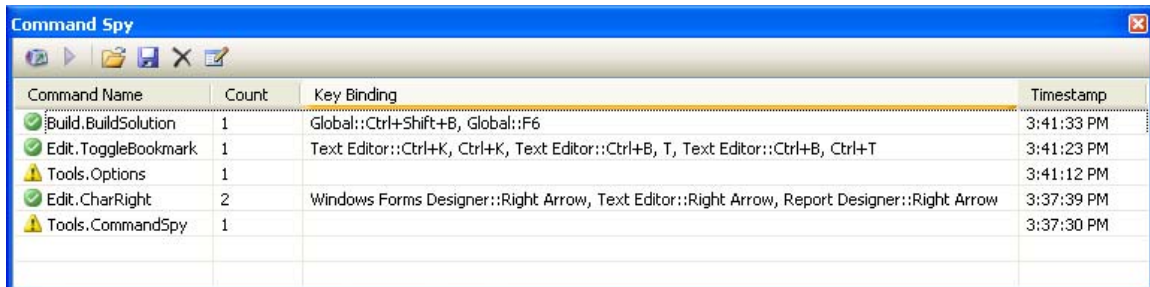
- By clicking on a menu item or toolbar button linked to that command
- By pressing a key sequence, such as **Ctrl+F**
- By typing it directly in the **Command Window**

Unfortunately, it’s almost impossible to tell what command is linked to which menu items or toolbar buttons. Sometimes you execute commands in Visual Studio and you don’t even know it. For instance, pressing the right arrow button in an editor invokes the **Edit.CharRight** command.

The **Command Spy** monitors command execution and allows you to see exactly what commands you’ve run, how many times you’ve run them and what key bindings are used to invoke those commands. The main purpose of this tool is to allow you to learn what commands are bound to which keystrokes, so that you can code faster within the IDE.

How does it work?

You can show the **Command Spy** by clicking **SlickEdit > Command Spy**. This will bring up the following tool window.



Command Name	Count	Key Binding	Timestamp
Build.BuildSolution	1	Global::Ctrl+Shift+B, Global::F6	3:41:33 PM
Edit.ToggleBookmark	1	Text Editor::Ctrl+K, Ctrl+K, Text Editor::Ctrl+B, T, Text Editor::Ctrl+B, Ctrl+T	3:41:23 PM
Tools.Options	1		3:41:12 PM
Edit.CharRight	2	Windows Forms Designer::Right Arrow, Text Editor::Right Arrow, Report Designer::Right Arrow	3:37:39 PM
Tools.CommandSpy	1		3:37:30 PM

Any time you execute a Visual Studio command, it shows up as an entry in the Command Spy. Each command has a single entry and when a command has been executed more than once, its count field is incremented. From the screen capture, you can see that five different commands have been executed and that **Edit.CharRight** has been executed twice. Each entry has the following fields:

- **Command Name** - The name of the command. This is the name that may be typed into the Command Window to execute the command.
- **Count** - The number of times the command has been executed while the Command Spy has been active.
- **Key Binding** - The key bindings that execute that command.
- **Timestamp** - The last time the command was executed.

Each entry also has an associated icon, which has the following meaning:

- ✓ The green Check Mark icon means that the command has a key binding.
- ⚠ The yellow Exclamation Mark icon means that the command doesn’t have a key binding.
- ✖ The red X icon means that the command doesn’t have a key binding **and** has been called ten or more times.

You can also sort any of the fields by clicking on their column headers. Click them multiple times to toggle between sorting in ascending and descending order. This allows you to see information

such as the most frequently executed commands (sort the **Count** column) or the last executed commands (sort the **Timestamp** field).

Again, the main purpose of this tool is to allow you to learn what commands are bound to which keystrokes, so that you can code faster within the IDE. Looking at the screen capture, it's clear that there is no key binding to open the Options dialog. If **Tools.Options** is a commonly executed command, then a key binding should be assigned to it so that you don't have to hunt for it in the menu whenever you want to open it. To learn how to assign key bindings to commands, please review the toolbar information below.

The toolbar

The **Command Spy** toolbar can be seen in the following screen capture.



1 2 3 4 5 6

1. **Go to key bindings** - Invokes the **Tools > Options** dialog and goes directly to the key bindings page. From here, you can bind a key combination to any command.
2. **Execute command** - Executes the currently selected command.
3. **Restore session data** - Allows you to load the data captured by the Command Spy from your last saved session.
4. **Save session data** - Allows you to save the current data captured by the Command Spy.
5. **Clear session data** - Clears all entries from the Command Spy.
6. **Show key binding summary file** - Creates a text file of the session data with all of the commands and key bindings. Printing this file is extremely useful in helping you to remember particularly useful key bindings.

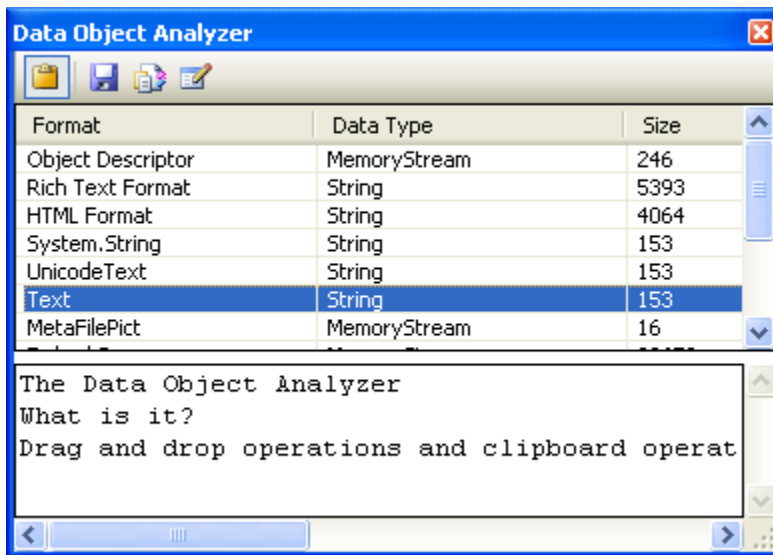
The Data Object Analyzer

Drag-and-drop operations and clipboard operations both work with objects that implement the IDataObject interface. Data objects contain [1...n] data items, which may be of any type, and are stored with an associated string-based key. There are some standards about what key/data pairs are used in clipboard or drag-and-drop operations, but for the most part, applications add whatever data they want. If you're writing an application that requires clipboard or drag-and-drop integration, it is important to understand what data is expected from other applications.

The Data Object Analyzer solves this problem by inspecting the contents of any clipboard operation, or drag-and-drop operation, from any other application. The Data Object Analyzer can then be used to automatically generate functions to handle any clipboard or drag-and-drop data.

How does it work?

You can show the Data Object Analyzer by clicking **SlickEdit > Data Object Analyzer**. This will bring up the following tool window.



The tool window is wired to inspect data objects in one of two ways:

1. **Copying anything to the clipboard** - The tool window registers itself for clipboard updates. Whenever any application copies something to the clipboard, the tool window automatically retrieves it and lists the data items. You can optionally turn off this behavior.
2. **Dragging and dropping** - The list view area of the tool window is a universal drop target. You can drag any item from any application and drop it onto the list view, and the tool window automatically enumerates the data items stored in the dropped item.

Once a data object is inspected, its data contents are displayed in the tool window's list view. Each entry in the list has the following fields:

- **Format** - The key (or name) used to retrieve the particular data item from the data object.
- **Data Type** - The data type of the object returned by the data object, given the format.
- **Size** - The size in bytes of the data.

Once the Data Object Analyzer lists the formats stored in the data object, there are several ways to view each individual data item.

1. Single-clicking on a line item will populate the lower pane with the associated data.
2. Double-clicking on a line item will open the data in a Visual Studio editor. You can also do this by selecting a line item and clicking the **View data in editor window** button.
3. You may select a line item and click the **Save selected data to file** toolbar button.

Auto-Generating Data Format Handlers

You may auto-generate a handler for any data format in the list view. To do this, open the code file that you want the handler to be added to. This code file must contain at least one class.

Select the data format you want to handle and click the **Generate handler for data format** toolbar button. A function will be added to the first class in the active code window. The function will take an IDataObject parameter and return the value of the selected format, given its key and data type. This provides a way to take an IDataObject reference and retrieve the data you need to handle. This functionality works with C#, VB.Net, and managed C++.

The toolbar

The **Data Object Analyzer** toolbar can be seen in the following screen capture.



1 2 3 4

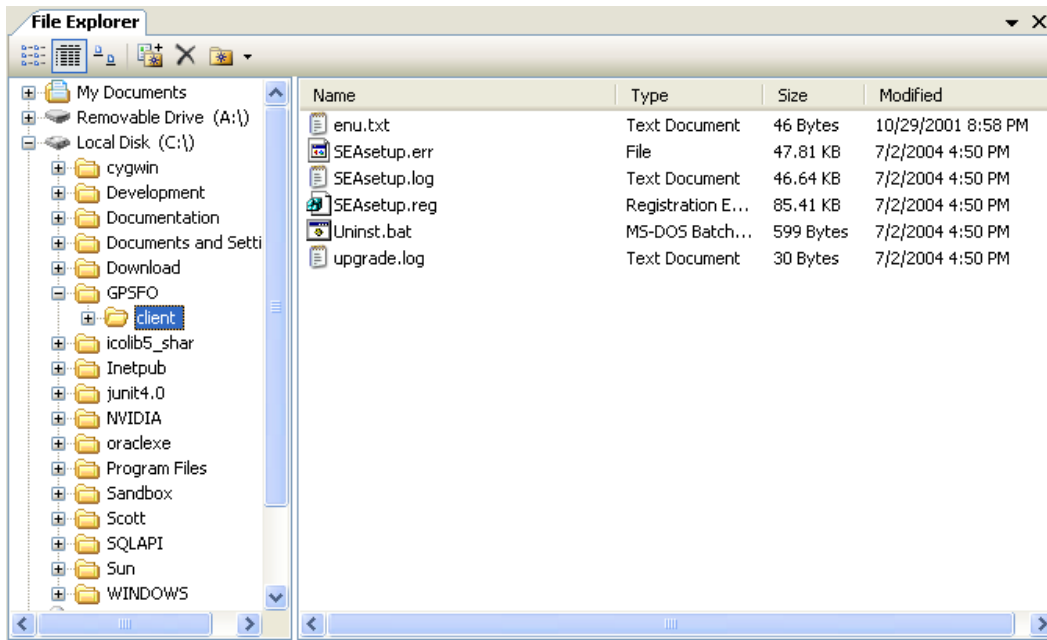
1. **Toggle listening for clipboard events** - Toggles whether or not the tool window should inspect clipboard data automatically. If the toolbar button is boxed (as shown in the toolbar screen capture), then it is active, otherwise it is not.
2. **Save selected data to file** - Opens the Save Data dialog and saves the selected data format to the selected file name.
3. **Generate handler for data format** - If a code editor is open and active, clicking this button will auto-generate a function to retrieve the selected data format from a data object.
4. **View data in editor window** - Creates a new editor window in Visual Studio and displays the contents of the selected data format. This is particularly useful for binary data items.

File Explorer

Using Visual Studio's Open File dialog to open files in Visual Studio can be very tedious, especially when you have to open files from several different directories. The **File Explorer** provides an easy way to open solutions, projects, or single files in Visual Studio. It also makes it easy to drag-and-drop files into an open Visual Studio project.

How does it work?

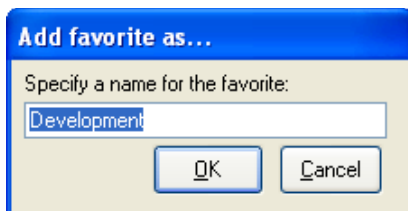
You can show **File Explorer** by clicking **SlickEdit > File Explorer**. This will bring up the following tool window. By default, the tool window is placed as a tabbed document.



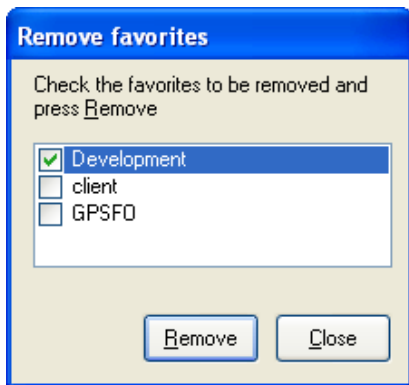
The interface is extremely similar to the **Windows® Explorer**. The file list pane may be sorted by any of the fields in ascending or descending order. Double clicking on a file will open it in Visual Studio. You may also drag-and-drop one or more files into the solution hierarchy to add them to any project.

Saving favorite folders

File Explorer allows you to save your favorite folders so that you can easily return to them. To do this, browse to one of your favorite folders, select it, and click the **Add favorite** toolbar button. You will be prompted with the following dialog:



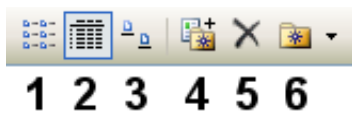
Enter a name for the favorite and click **OK** to add it. The favorite will be added to the Favorites drop-down list on the toolbar and may be navigated to at any time. Favorites may be removed by clicking the **Remove favorites** toolbar button. You will be prompted with the following dialog:



Check the favorites that you want to remove and click the **Remove** button to take them out of the favorites drop-down list.

The toolbar

The **File Explorer** toolbar can be seen in the following screen capture.



1. **List view** - Lists the files in small icon list mode.
2. **Details view** - Lists the files with associated file details.
3. **Icon view** - Lists the files in large icon list mode.
4. **Add favorite** - Adds a favorite entry to the currently selected folder.
5. **Remove favorites** - Allows you to remove one or more favorites.
6. **Favorites drop-down list** - A drop-down list of all stored favorites.

Editor Gadgets

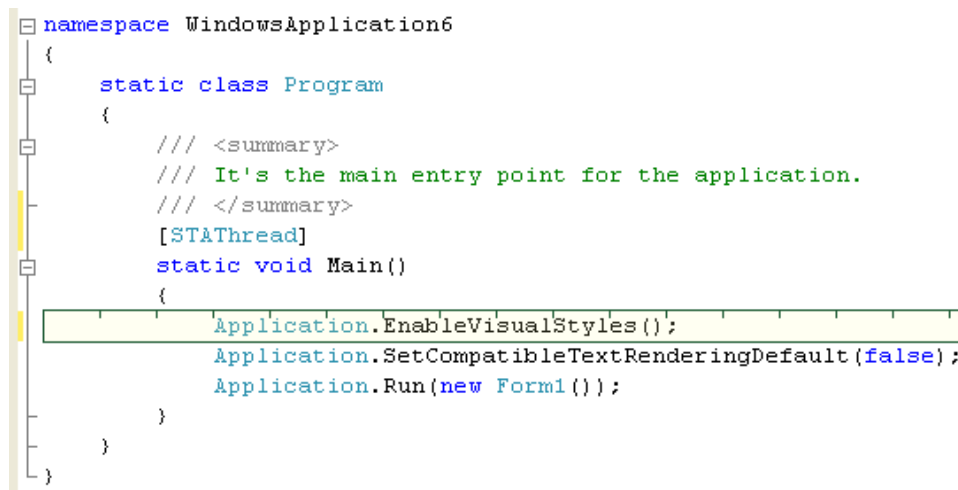
Editor Gadgets is a collection of four small utilities to add spice to your Visual Studio editor. These gadgets include:

- **Line ruler** - Places a “ruler” highlight across your current line (where the cursor is) and has tick marks to indicate the editor’s indentation levels.
- **Indentation guide** – Draws a vertical bar in the editor indicating the indentation level of the current line.
- **Auto-copy selection** - Automatically copies text to the clipboard when selected with the mouse. Paste may be done by simply clicking in the editor with the middle mouse button (typically the scroll wheel on most mice). These behaviors are similar to XMouse.
- **Editor Graphic** - Allows you to add a graphic to your editor, either as a single image or as a tiled background.

How does it work?

Line ruler

The line ruler is shown in the following screen capture:



```

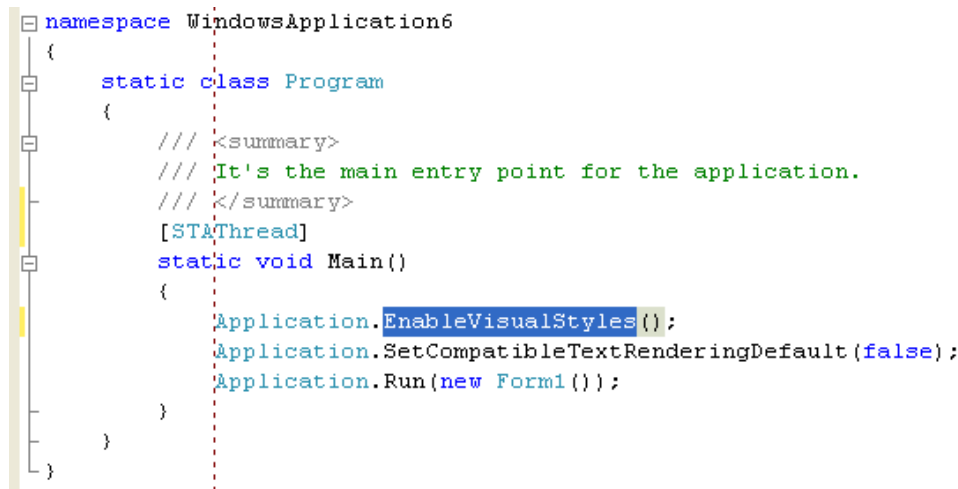
namespace WindowsApplication6
{
    static class Program
    {
        /// <summary>
        /// It's the main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

The line ruler highlights your current line. It optionally draws indentation tick marks which provide an easy way to see where the indentation levels are in your source code. The line ruler may be turned on or off, and may be configured using the **SlickEdit > Editor Gadgets** options page.

Indentation guide

The indentation guide is shown in the following screen capture:



```

namespace WindowsApplication6
{
    static class Program
    {
        /// <summary>
        /// It's the main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

The indentation guide draws a vertical dashed bar in the editor indicating the indentation level of the current line. The indentation guide always stays visible no matter how far away from the current line you scroll, so it provides an excellent way to match indentation levels for lines that are too far apart to view on the screen at one time. The indentation guide may be turned on or off, and may be configured using the **SlickEdit > Editor Gadgets** options page.

Auto-copy selection

When auto-copy selection is active, any selection made with the mouse will automatically be copied to the clipboard. An acknowledgement will appear in Visual Studio's status bar with the text, "Copied text to clipboard." Copied text may be pasted anywhere by simply clicking the middle mouse button. Auto-copy selection may be turned on or off using the **SlickEdit > Editor Gadgets** options page.

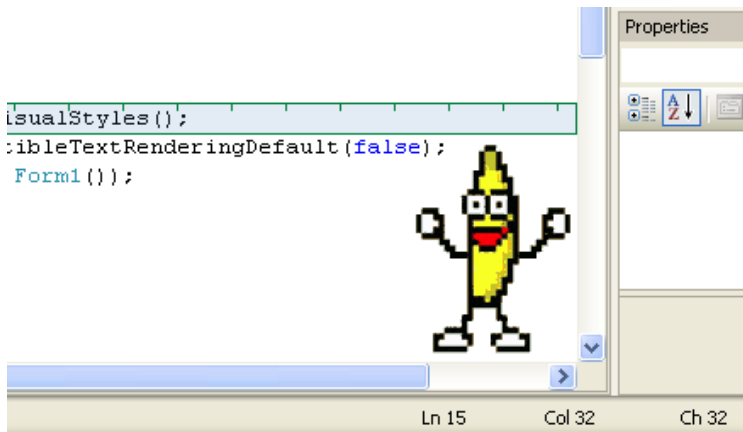
Editor graphics

Editor graphics fall into two categories:

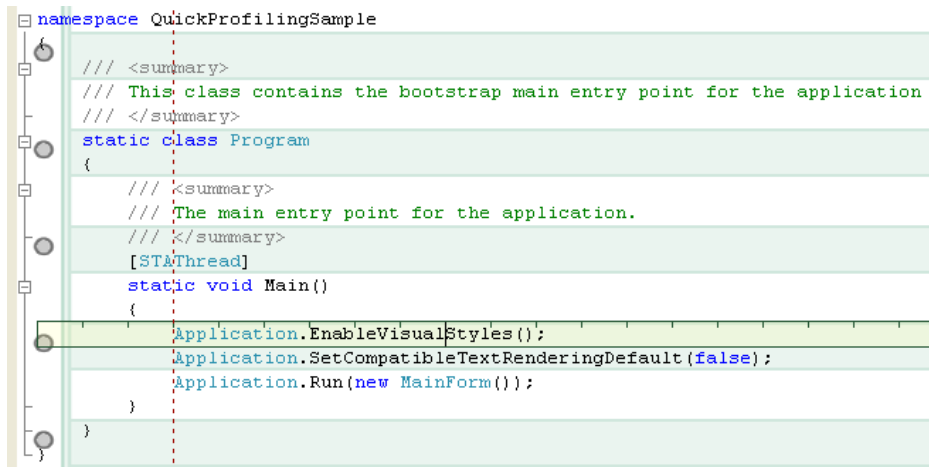
- **Single graphic images** - These graphics are rendered once in the editor window. They may be placed in any of eight different locations around the edge of your editor window.
- **Repeating graphic tiles** - This is basically "wallpapering" your editor. The selected graphic will be repeated vertically and horizontally to fill the editor window.

All editor graphics may be locked in place or may be allowed to scroll within the editor. They may also be rendered opaque or with any degree of transparency. This feature may be turned on or off, and may be configured using the **SlickEdit > Editor Gadgets** options page.

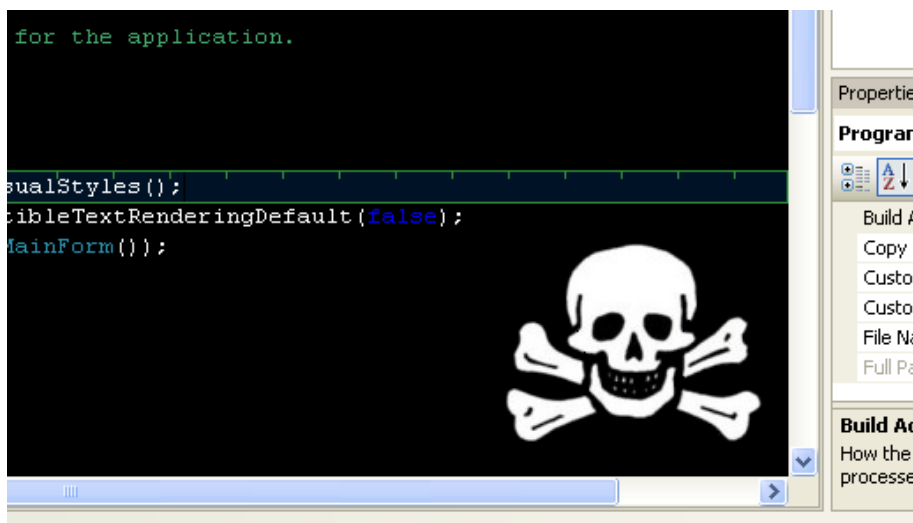
The following screen capture shows a single graphic image that's been locked into the bottom right corner of the editor.



The following screen capture shows a tiled green bar paper pattern. Repeating image tiles work well with subtle graphics and high transparency, which can be configured in the options page.

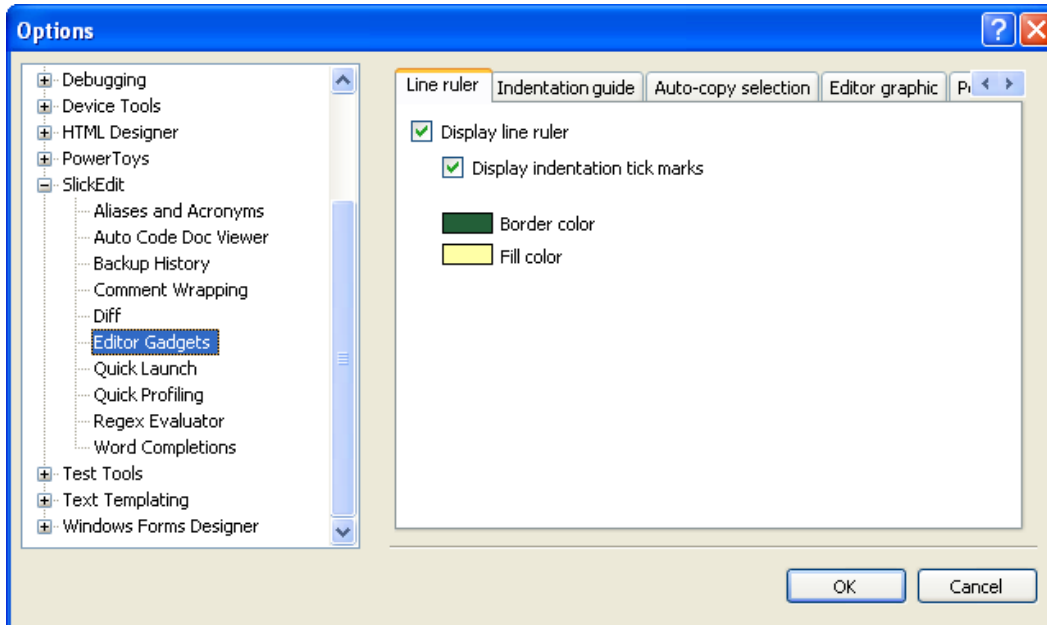


Adding a graphic or ruler bar can be configured to look good for developers with black backgrounds as well, as shown in the following screen capture.



Configuration

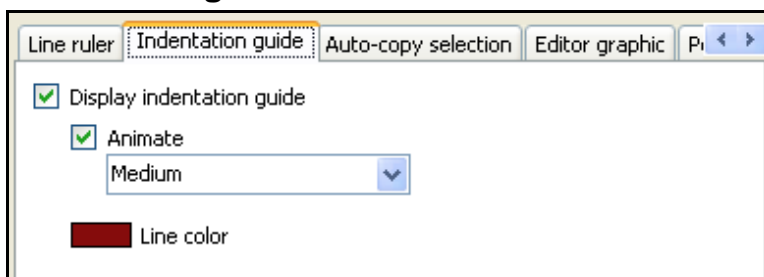
Editor Gadgets can be configured by selecting **Tools > Options** on the main menu and going to **SlickEdit > Editor Gadgets** in the options dialog tree. There are four tabs, one for each editor gadget.



Line ruler

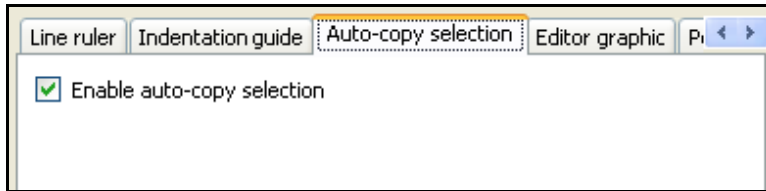
- **Display line ruler** - Sets whether or not the line ruler should be drawn. If this option is checked, then the border and fill colors of the ruler may be set by clicking on their respective color boxes.
- **Display indentation tick marks** - Sets whether or not indentation tick marks should be drawn.

Indentation guide



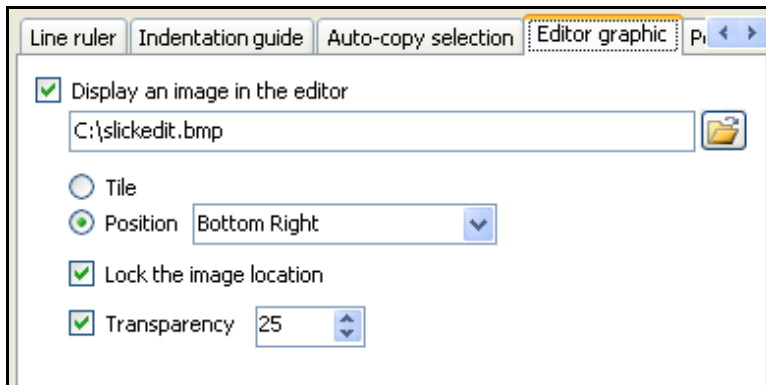
- **Display indentation guide** - Sets whether or not the indentation guide should be drawn. If this option is checked, then the line color of the guide may be set by clicking on the Line color box.
- **Animate** - Sets whether or not the indentation guide should be animated. Animating the guide makes it move in a sweeping motion instead of just snapping to its new location. If this option is set, you may also set the speed drop-down box to a speed that looks best to you.

Auto-copy selection



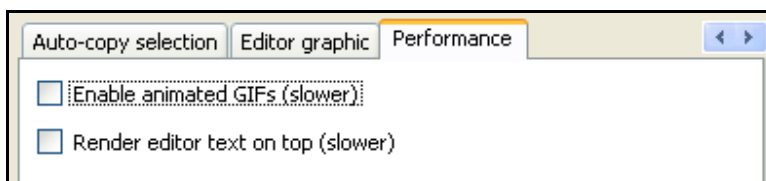
- **Enable auto-copy selection** - Sets whether or not the auto-copy selection feature should be enabled.

Editor Graphic



- **Display an image in the editor** - Sets whether or not the editor graphics feature should be enabled. If checked, you may click the “Browse” button to select a graphic file to display. See the “Graphic Recommendations” section for more information on selecting graphic files.
 - **Tile / Position** - If the Tile option is selected, then the selected graphic will be tiled across the whole editor. Otherwise, if the Position option is checked, then the graphic will be placed as selected in the Location drop-down list box.
 - **Lock the image location** - If checked, then the graphic will not be affected by scrolling in the editor. If unchecked, then the graphic will move as you scroll the editor.
 - **Transparency** - If checked, this option allows you to set the percentage of transparency for the graphic. Unchecking this option renders the graphic opaque. Valid transparency values are in the range of 5 through 95. The greater the value, the more transparent the graphic will be.

Performance



- **Enable animated GIFs** - Sets whether or not GIF images with multiple frames should be animated. Animating GIF images requires more processing time. This option has no effect if the selected image is not a multi-frame GIF.
- **Render editor text on top** - Sets whether or not the editor text should be rendered on top of the editor gadgets or underneath. This option will have the most effect if you are tiling a bitmap in the editor. Rendering the editor text on top requires more processing time.

Graphic Recommendations

The following recommendations will help you get the most from the editor graphics feature:

- The top left pixel is used to determine which color to make transparent.
- Performance will be enhanced by setting the graphic to a lower color depth.
- BMP, JPG, GIF, and PNG formats are supported.
- If you are tiling a graphic, it's recommended that you set the transparency to somewhere around 85% and render the editor text on top (see the **Performance** options tab).

The SLOC Report

This tool provides an easy way to count the lines of code in a solution, project, or in an individual file. The line count is divided into three categories: code, comments, or whitespace. Once the lines of code have been counted, the results are drawn as a pie graph.

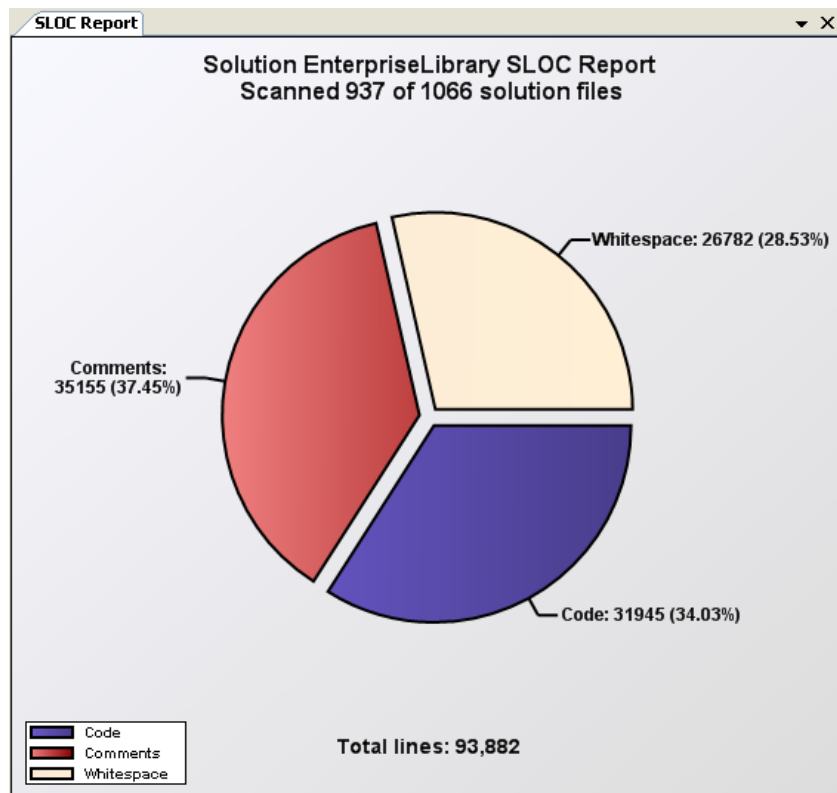
How does it work?

SLOC stands for Source Lines of Code. http://en.wikipedia.org/wiki/Source_lines_of_code
To generate a SLOC report, right click on any item in the Solution Explorer and select the **Count lines of code** command. If the selected item is a solution, then all lines of code in the solution will be counted. If the selected item is a project, then all lines of code in the project will be counted. If the selected item is a single file, then all lines of code in the file will be counted.

The report counts SLOC using the following strategy:

- If a line is only a single line comment, it counts as a comment line
- If a line is part of a multi-line comment, it counts as a comment line
- If a line has one or less characters, then it is whitespace (this accounts for brace lines)
- If a line has two or more non-comment characters, it is a line of source code

The following screen capture shows a SLOC report for the Enterprise Library.



SLOC reports can count lines of code in VB, C#, J#, and C++. It does not parse HTML, XML, or ASPX files.