TME285
Intelligent Agents

Tutorial 1:
Basic GUI components and event handlers

Mattias Wahde
2018

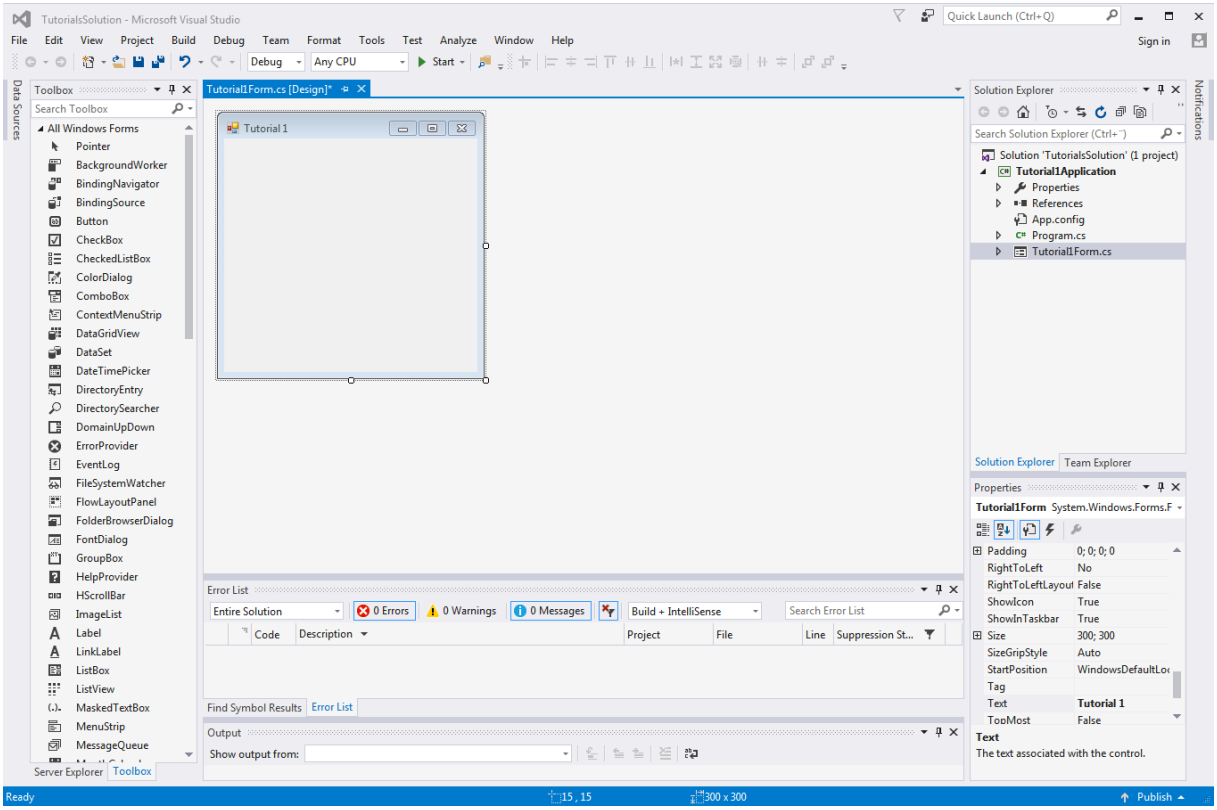

Fig. 1 Opening Visual Studio.

1. Introduction

In this tutorial, we shall write some basic C# code, focusing on designing a graphical user interface (GUI) for a C# application, and learning how to respond to user actions.

Note: The exact appearance of the Visual Studio environment may depend somewhat on the version used. The tutorial has been generated using VS2015. However, the tutorial should be easy to follow regardless of the version used.

1.1 Preparation

Before proceeding read Section A.1 in the compendium!

Notes:

(1) It is important that you follow the instructions closely – they are given for your benefit!
(2) The tutorials are quite similar (in the level of complexity) to the code examples in the Examples.zip file, but they are not identical – you should read Appendix A, go through the code in the Examples.zip file, and complete all the tutorials.

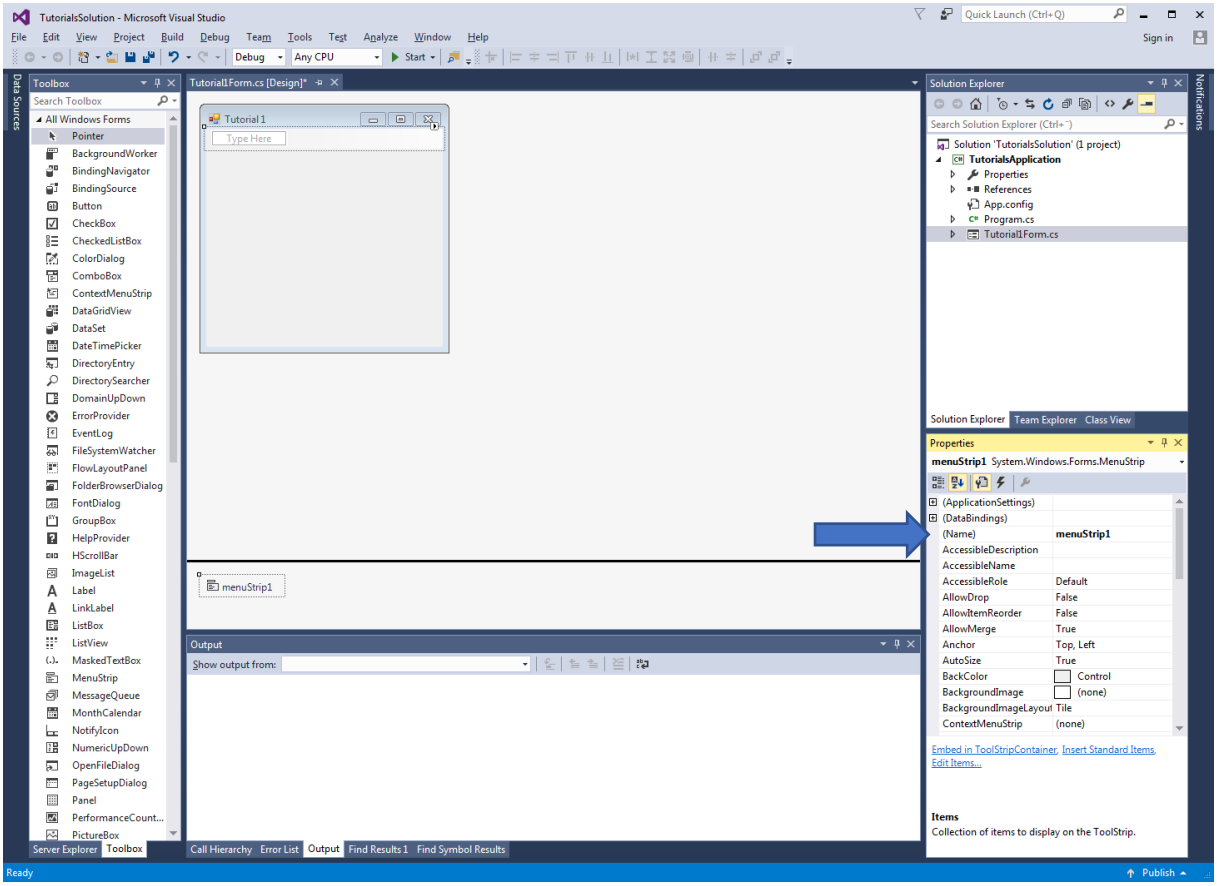

Fig. 2: Adding a menu strip. The blue arrow has been added to indicate the Name property.

2. Step-by-step tutorial

- Extract the TutorialSolution.zip file and double-click on TutorialSolution.sln to start Visual studio. The window that appears should look (roughly) as in Fig. 1 above.

- If you do not see the small quadratic form (window) with the title “Tutorial 1”, then double-click on the “Tutorial1Form.cs” in the Solution Explorer.

- Move the mouse pointer to the Toolbox (on the left-hand side of the window), grab a MenuStrip by holding down the mouse button over the MenuStrip, and drag it to the form (i.e. the small, quadratic window entitled “Tutorial 1” in the Windows Forms Designer (see Appendix A.1). Release the mouse button. A menu strip now appears on the form, see Fig. 2.

  As can be seen in the Properties window, the name of the menu strip is menuStrip1. C# will generally use this naming convention by default (i.e. component name plus a number), but we are free to change the name.

- Find the Name property (listed as “(Name)”) in the properties window), indicated with a blue arrow in Fig. 2. Change the name to “mainMenuStrip”, by clicking (once) on the text menuStrip1, and then editing the text as just described. This action will not cause any visible change, but it is a good habit to use appropriate, descriptive variable names. rather than the default names assigned by Visual Studio. Now, the purpose of the menu is, of course, to handle some basic user actions, and one should follow the normal practice of Windows programming, placing (for example) the File-related menu items to the left etc

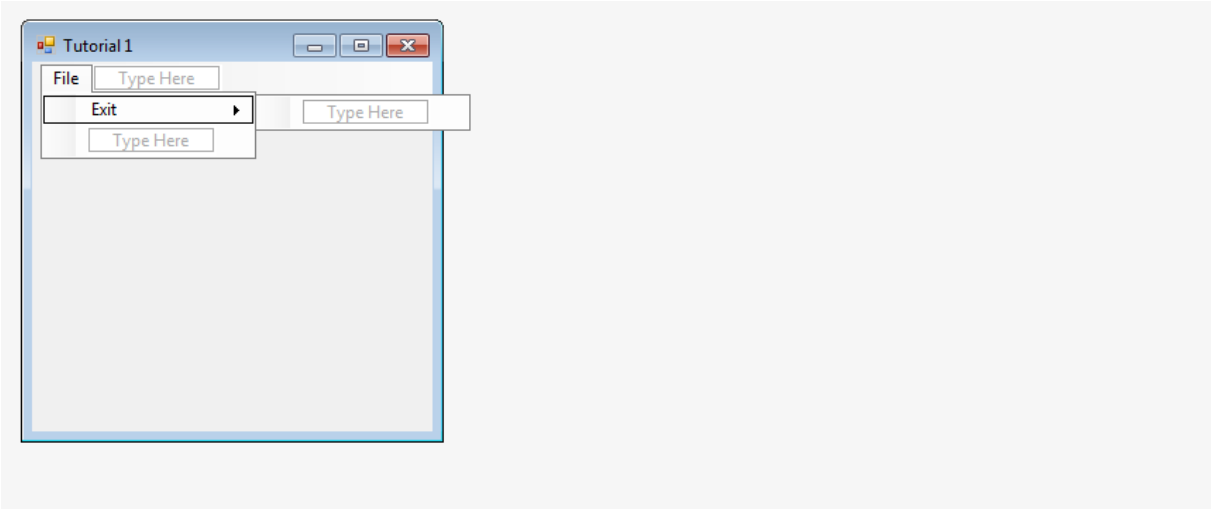

Fig. 3. Adding menu items.

- In the form, where it says “Type here”, write “File”. Then, just below this text, add the text “Exit”. Do *not* click on the menu items just added. The form should now look as in Fig. 3.

- Now, double-click on the Exit menu item. You will now be taken to the code associated with the form that we are designing. When one double-clicks on a graphical component, C# will automatically write skeleton code for an **event handler** associated with the user action. However, note that every graphical component has several events, and the one selected by C# (upon a double-click in the designer) many not always be the relevant one. Below, we will see how one can add an event handler directly in the code, rather than double-clicking on the GUI.

- Now, in this method (an event handler is a special case of a **method**), click on the Code Editor, (see also Appendix A.1) and add the text “Application.Exit()”. As is rather evident, this code will cause the program to stop execution. The parentheses () are needed to indicate that Exit is itself a method, taking no inputs. The code should now look as in Fig. 4.

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace TutorialsApplication
12  {
13      public partial class Tutorial1Form : Form
14      {
15          public Tutorial1Form()
16          {
17              InitializeComponent();
18          }
19
20          private void exitToolStripMenuItem_Click(object sender, EventArgs e)
21          {
22              Application.Exit();
23          }
24      }
25  }
26
```

Fig. 4. The code after adding the call to the application’s Exit command.

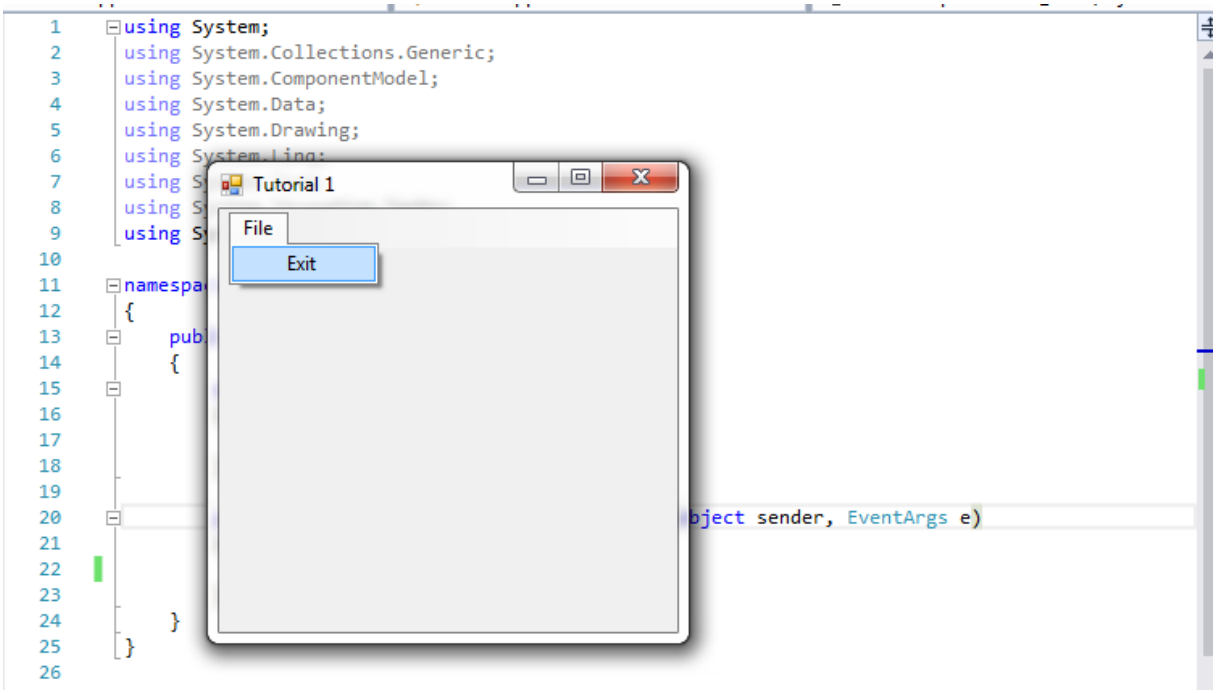

Fig. 5. Running the program.

- At this point, we have a functional, if simple, C# program. To Run the program either (i) click on the green arrow with the text "Start" at the top of the Visual Studio window or (ii) click F5.

  Now the form appears, with the File menu.

- Click once on the File menu, so that the Exit menu item appears. Then click (once) on Exit (as you would do when exiting any Windows program), as shown in Fig. 5. As you can see, the form disappears, and the program stops execution: You are taken back to the Code editor. The next step is to add some actions to the program (rather than just the ability to exit)

- Note that one can easily switch between code editing and GUI design. At this point, you will see the code editor. In order to view the form, click on the tab "Tutorial1Form.cs [Design]" (at the top). Alternatively, you can right-click on the Tutorial1Form.cs in the Solution Explorer (top right in Visual Studio), and then select View Designer (to see the GUI) or View Code (to see the underlying code); see also Appendix A.1.

- When you get back to the Windows Forms Designer, click once on the form (anywhere except on the menus) to hide the menus. Then drag a button from the toolbox onto the form (hold the mouse button down while dragging). Place the button near the upper left corner, just below the menu. In the Windows Forms designer, you can easily move components about by dragging them with the mouse. The form should now look as in Fig. 6.

- Next, click once (note! do **NOT** double-click) on the button, and then change its name (the "(Name)" property) to setBlueButton and the Text to "Set blue" in the Properties window.

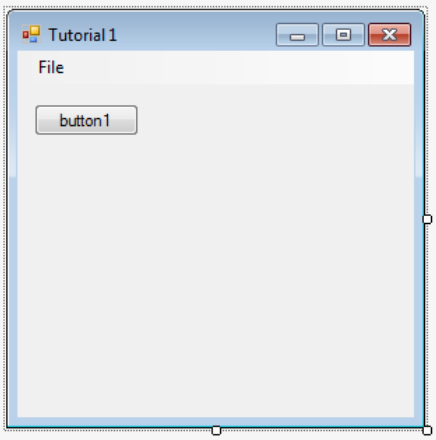

Fig. 6. Adding a button.

- We shall now add an event handler in a different way (compared to the Exit event handler added above). If you accidentally double-clicked on the button, C# will write the skeleton code as described above. However, if you didn't (i.e. if you followed the instructions…) you can now instead click on the little lightning bolt, in the Properties window, as indicated in Fig. 7.

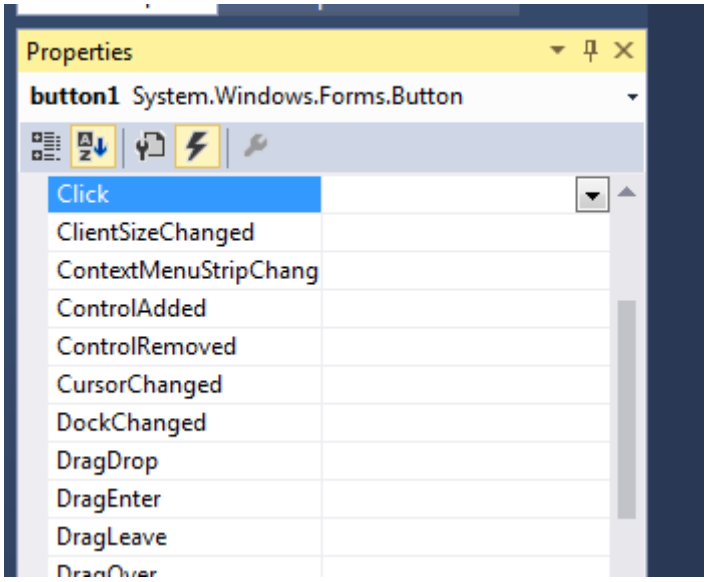

Fig. 7. Clicking on the lightning bolt to see the list of events associated with the button.

- Now, double-click on the highlighted text "Click" (see Fig. 7) in the Properties window. As you can see, C# automatically writes skeleton code for the click event. However, as you can see in Fig. 7, there are many other events associated with the button. In this particular case, since we want to write the Click event handler, it would have been possible just to double-click on the button in the Windows Forms designer, but if we wanted to add some other event handler (for example ClientSizeChanged), we should use the approach just described (or add the event handler manually, as described below).

- Next, in the Click event handler, add the code "this.BackColor = Color.Blue". Here, "this" refers to the form which we are editing. Since there is not risk of confusion, one does not actually need the "this" here, but there can be cases where one must include it. Note: If one

instead wanted to change the back color of the button, rather than the form, one should write “setBlueButton.BackColor = Color.Blue.

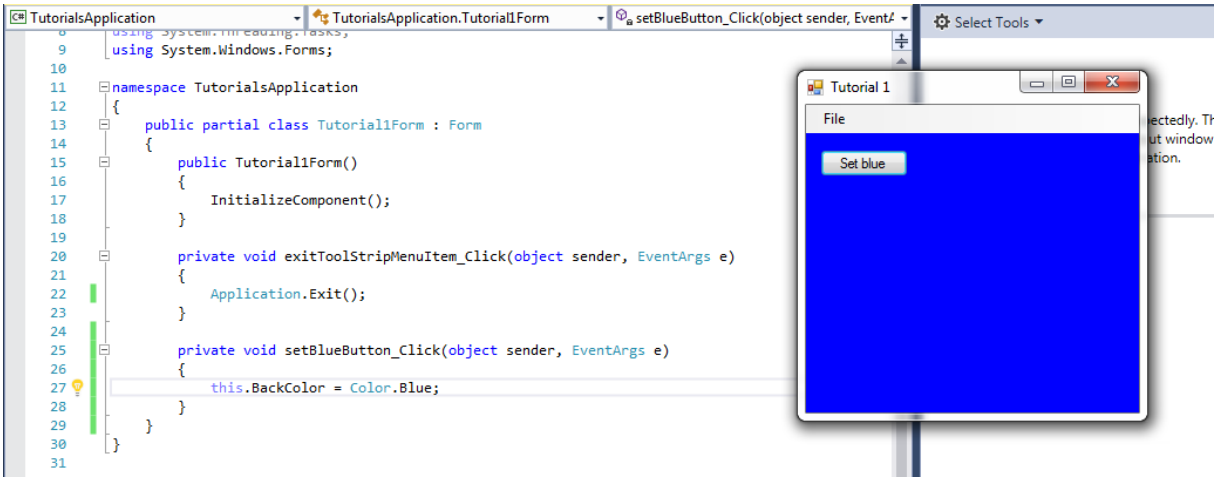

Fig 8: Running the program.

- Now run the program and click on the button. As can be seen, the back color of the form is changed to blue, as shown in Fig. 8. Next, select the Exit menu item to exit the program.

- To end this basic tutorial, we shall show yet another way to add an event handler. Start by adding another button, changing its name to setGreenButton, and its Text property to “Set green”.

  Note: When you have just added the new button, the Properties window will probably be in the event mode (as we just added the event handler for the setBlueButton). To change back to the Properties view, click on the small icon (with a picture of a wrench in front of a paper) just to the left of the lightning bolt; see also Fig. 7 above.

- Next, open the code view, either by clicking on the “Tutorial1Form.cs” tab at the top or right-clicking on the Tutorial1Form.cs in the Solution Explorer and selecting “View Code”. As you can see, no code has been added to handle the click event for the setGreenButton. Now, in the constructor for the form, i.e. the method called Tutorial1Form(), at the top, after InitializeComponent() and the following line of text:

```
setGreenButton.Click += new EventHandler(HandleSetGreenButtonClick);
```

- This line of code tells C# that it should add, to the Click event of the setGreenButton, a method (an event handler) called “HandleSetGreenButtonClick”. Of course, we have not written that method yet, so C# cannot find it, and it is therefore underlined with a red, wavy line.

- Next, still in the code editor, below the event handler for the other button (the setBlueButton), add the following method

```
private void HandleSetGreenButtonClick(object sender, EventArgs e)
{
    this.BackColor = Color.Green;
}
```

- As should be clear by now, this method will set the back color of the form to green. Now run the program once more, and try clicking the setGreenButton. The entire code for the tutorial is shown in Fig. 9 below.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TutorialsApplication
{
    public partial class Tutorial1Form : Form
    {
        public Tutorial1Form()
        {
            InitializeComponent();
            setGreenButton.Click += new EventHandler(HandleSetGreenButtonClick);
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void setBlueButton_Click(object sender, EventArgs e)
        {
            this.BackColor = Color.Blue;
        }

        private void HandleSetGreenButtonClick(object sender, EventArgs e)
        {
            this.BackColor = Color.Green;
        }
    }
}
```

Fig. 9 The complete code for Tutorial 1.

3. Final notes

Some more things to note:

- An event handler takes two inputs: (1) the sender, which for the button click events is the respective button, and for the exit event is the exit menu item, (2) an instance of EventArgs, which can be used to transfer information about the event in question; see also Appendix A.6.
- There are many different GUI components. Try adding (1) a textBox with an appropriate event handler and (2) a Label, with some text that is changed when the user clicks one of the buttons.
- Of course, an event handler can be arbitrarily complex, consisting of many lines of code (not just one line as in the examples above).
- **Removing an event handler added by accident**: If one accidentally double-clicks on the Windows Form Designer, an unwanted event handler may be added. For example, try double-clicking directly on the form (i.e. not on the buttons, the menu strip, or the frame). In

that case skeleton code for the form's Load event handler (which we do not need here) is automatically written. Now, one cannot just remove the corresponding method from the code editor, since the form now expects to find that method. Instead, in order to remove it, one must open the Properties window (for the form), click on the little lightning bolt to view the associated events, scroll down to the event in question (the Load event in this case), and remove the coupling between the event ("Load") and the corresponding code, by simply removing the text (i.e. deleting the text "Tutorial1Form_Load", in this case). When one does so, the method will also disappear from the code editor (if the method is empty, that is).