# ENG 1D04 Lab 1: Introduction to C#, Visual Programming and Event-Driven Architectures Term 1 2009/2010

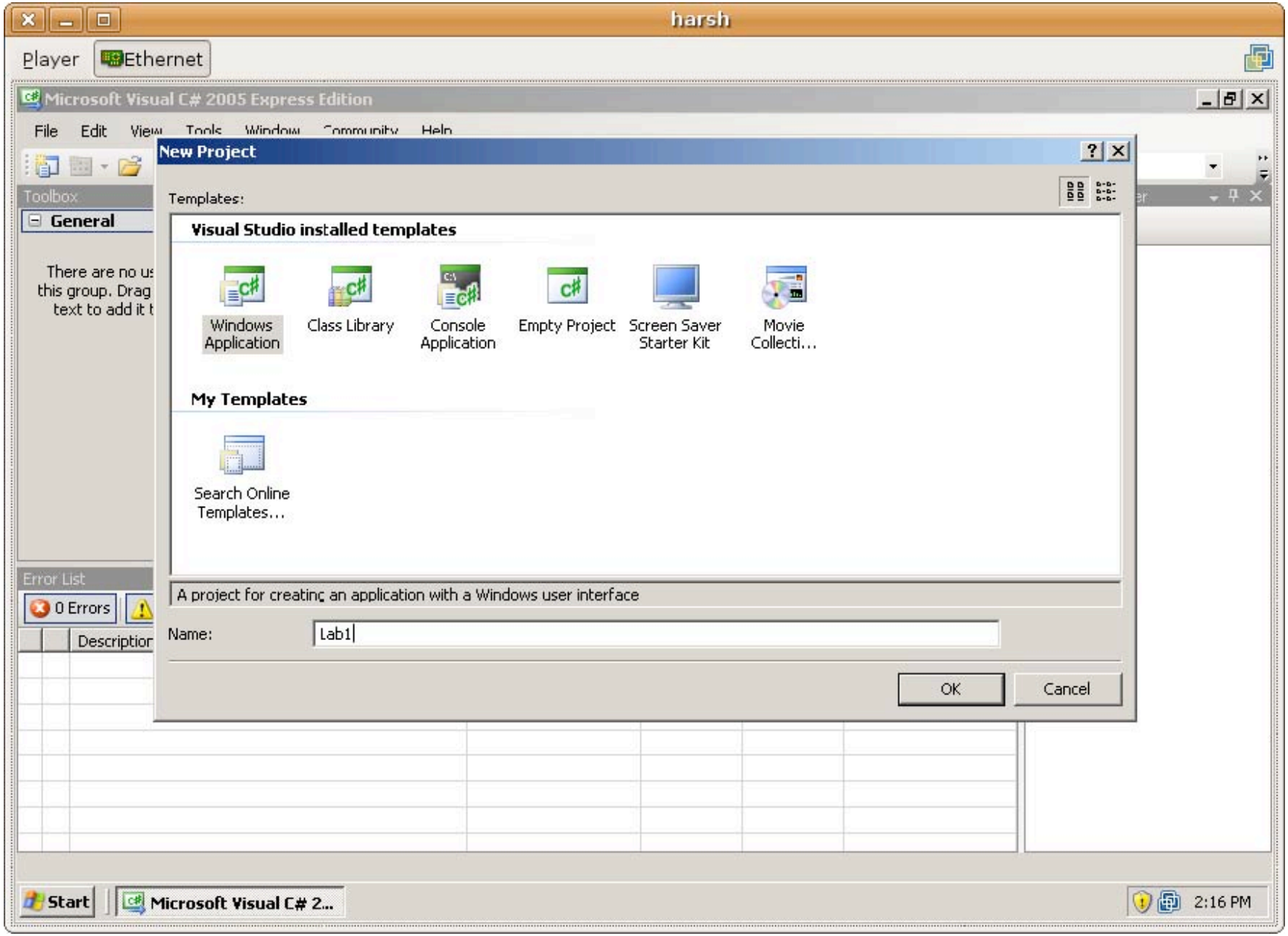
The purpose of the first lab is to demonstrate how to use the Visual Studio development environment to write a simple GUI (graphical user interface) application. The lab stresses the idea of an event-driven architecture. The goals of the lab are to understand how to create a simple user interface using the Visual Studio form designer and to understand the concept of an event as a user action that triggers execution of a named code segment.

Start by logging in.

Find Visual Studio on the Start menu and run it.

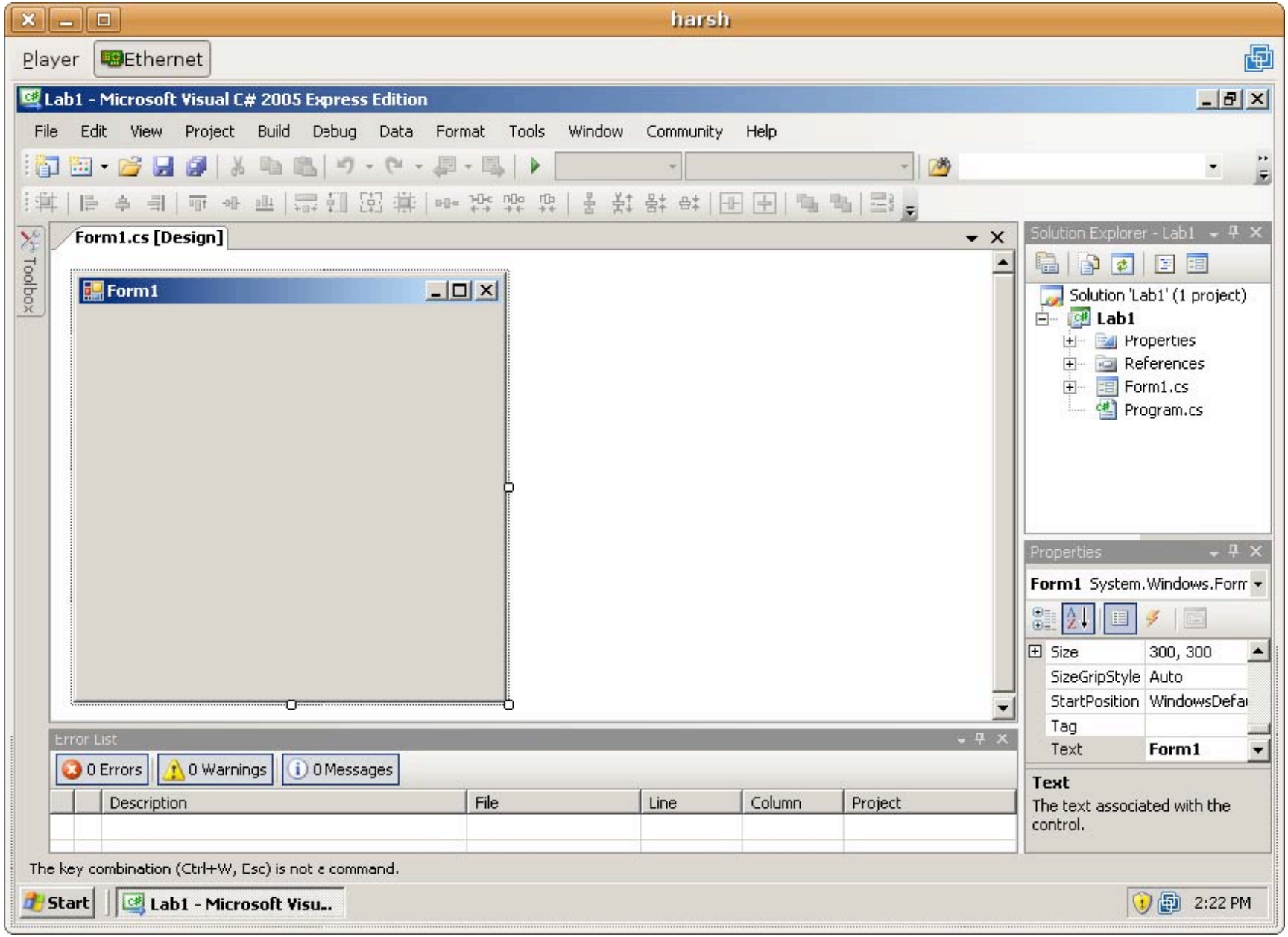
When it starts, you want to create a new project. This can be done by clicking on the "new project" icon (far left on the toolbar).

You will see a dialog similar to the following:

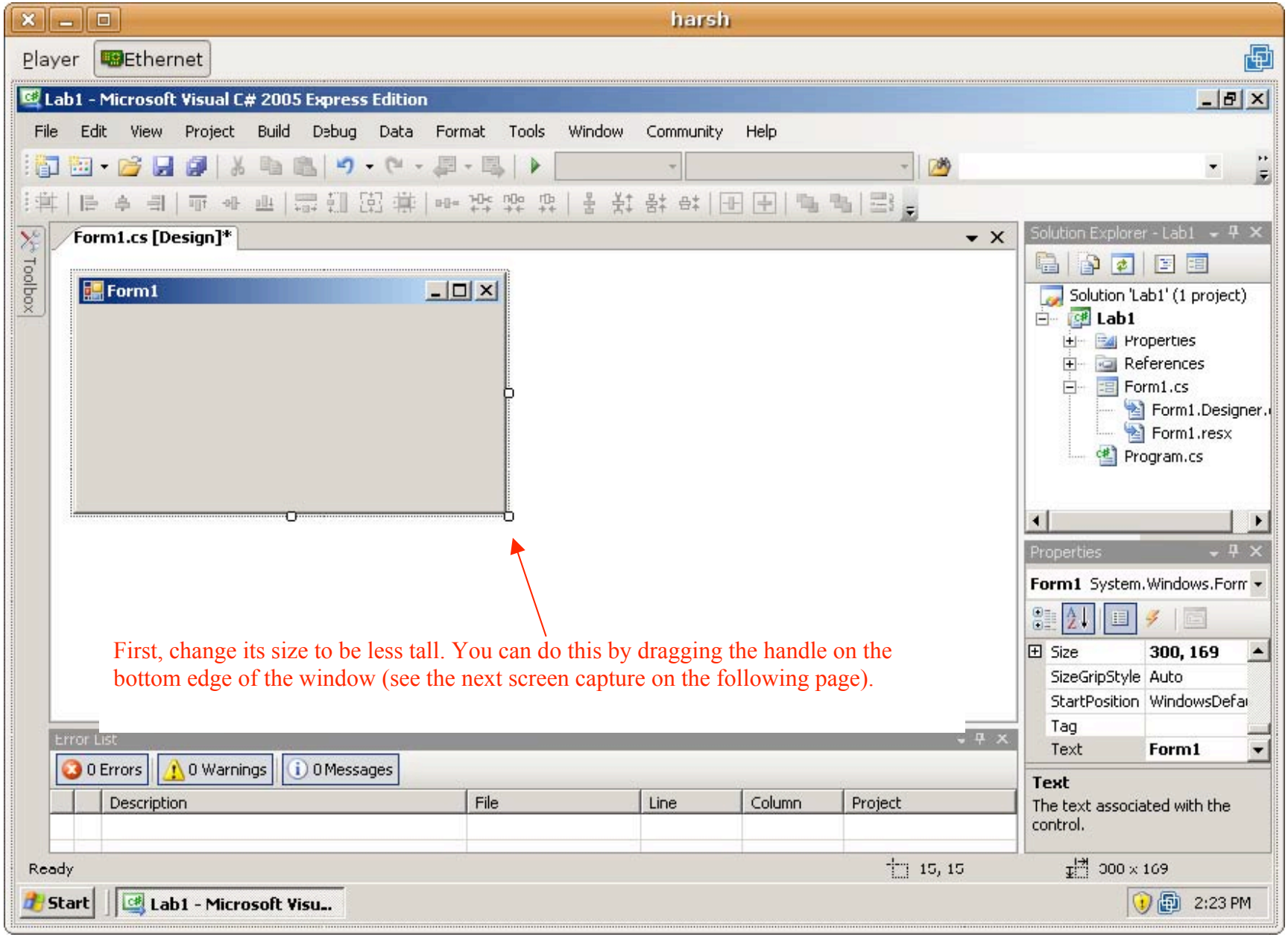



Create a "Windows Application" and name it Lab1.

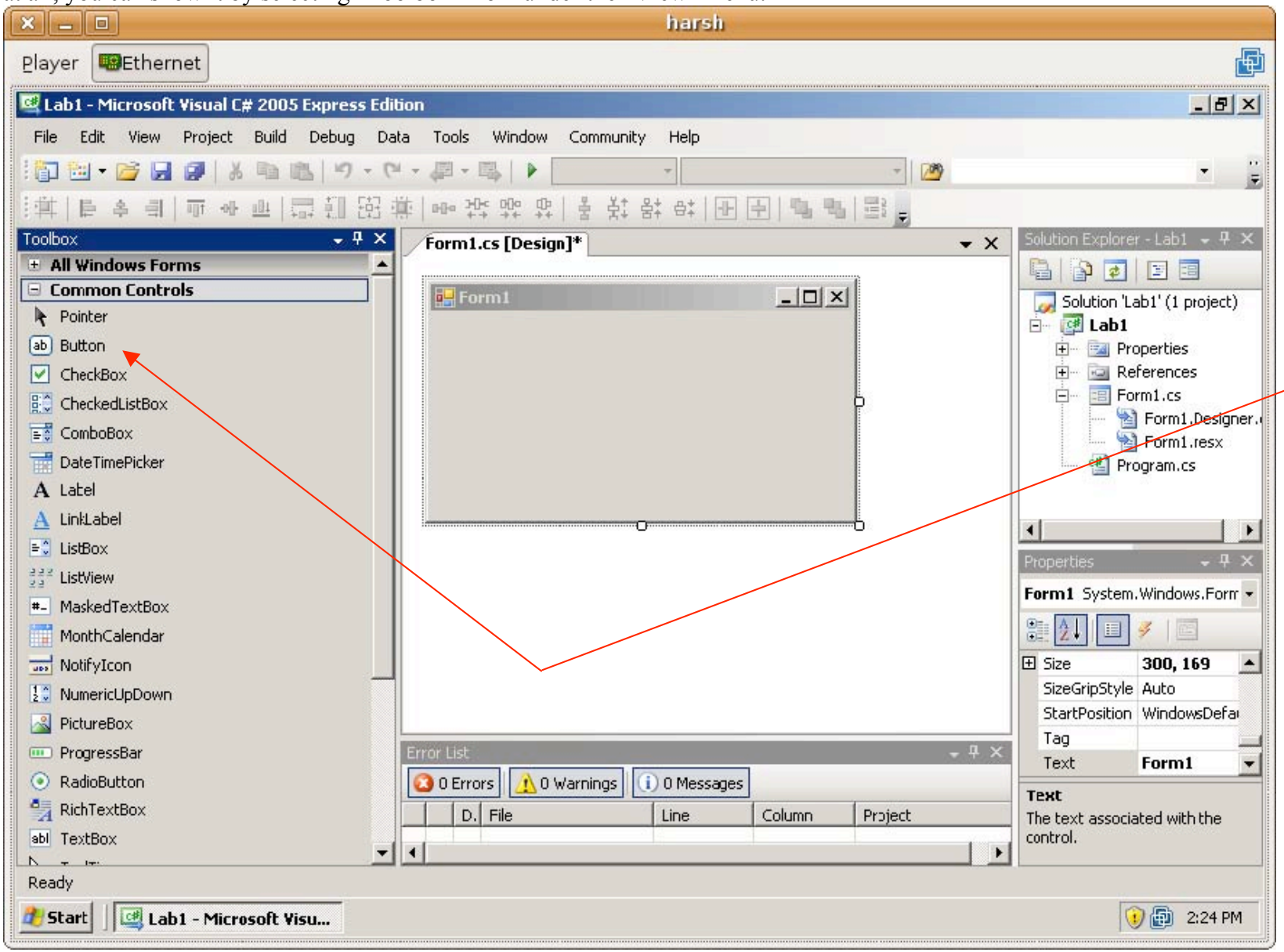
Once you click "OK" the following screen will be shown:



There may be some differences in configuration (such as placement of the tool bars and boxes). A new windows application will start with an empty form called Form1. This form represents the user interface that will be shown when the application is first run. You can interact with it to change its appearance.



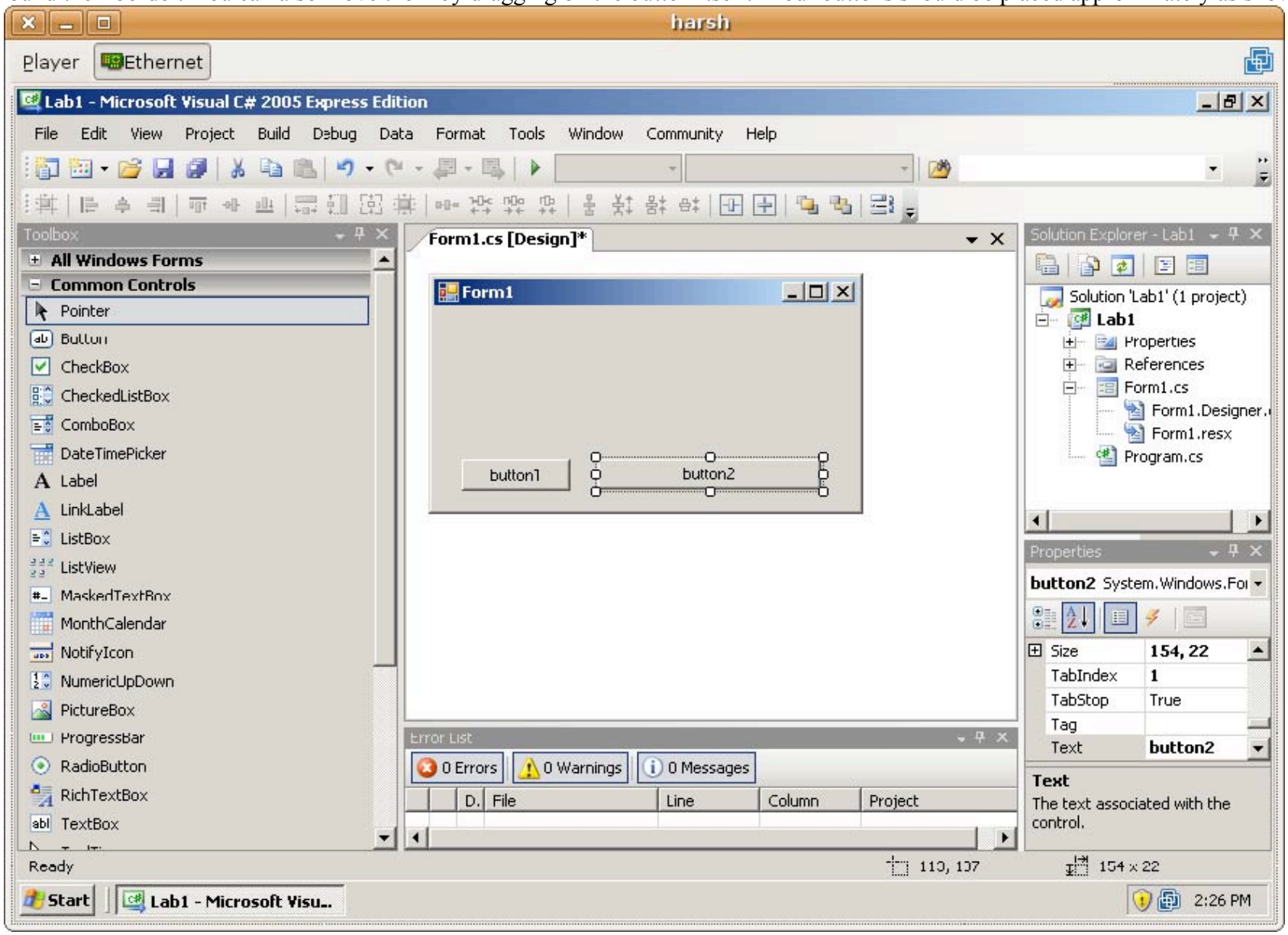
In order for an application to be useful it will need to have widgets for the user to interact with. Widgets are items like buttons, scrollbars icons and text entry boxes. In order to add widgets to the application, the "Toolbox" window must be visible. If there is "Toolbox" item to the left of the display hovering the mouse pointer over it should make the toolbox visible. You can then prevent it from auto-hiding by clicking on the "pin" icon. If you cannot find the toolbox at all, you can show it by selecting "Toolbox" from under the "View" menu.



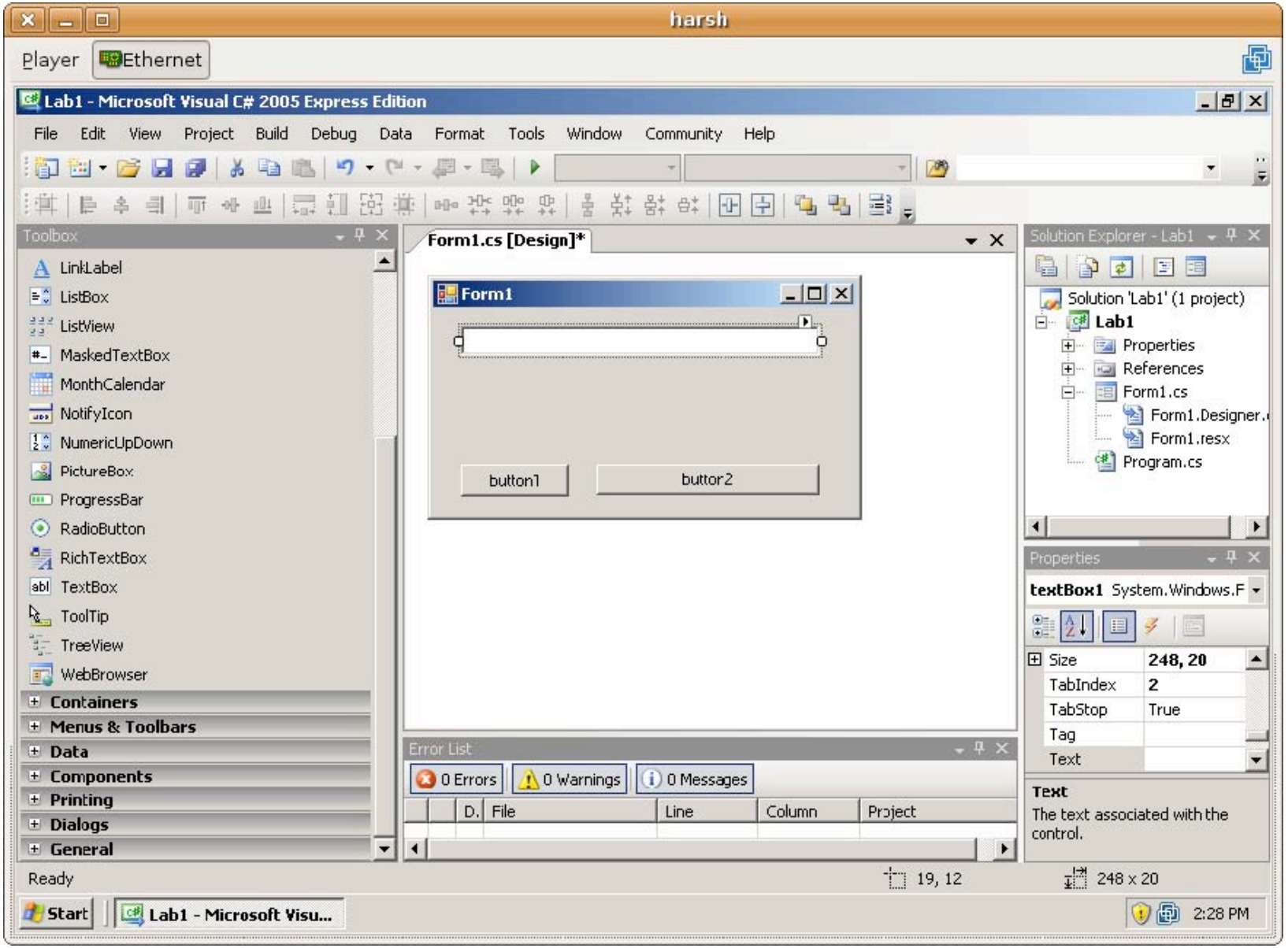

The toolbox has a number of categories inside of it. All of the widgets used in this lab are under the "Common Controls" category. Click the "+" icon beside the category to view its contents.

Select "Button" from under "Common Controls" and then click on the bottom left of the form to add a button. The button will be created with the name "button1" and the default shape and size. You can create buttons with a different shape and size by dragging an outline instead of just clicking.

Select "Button" again and this time drag the outline of the second button. After placing buttons on the form you can resize them by dragging on one of the handles around their border. You can also move them by dragging on the button itself. Your buttons should be placed approximately as shown below.

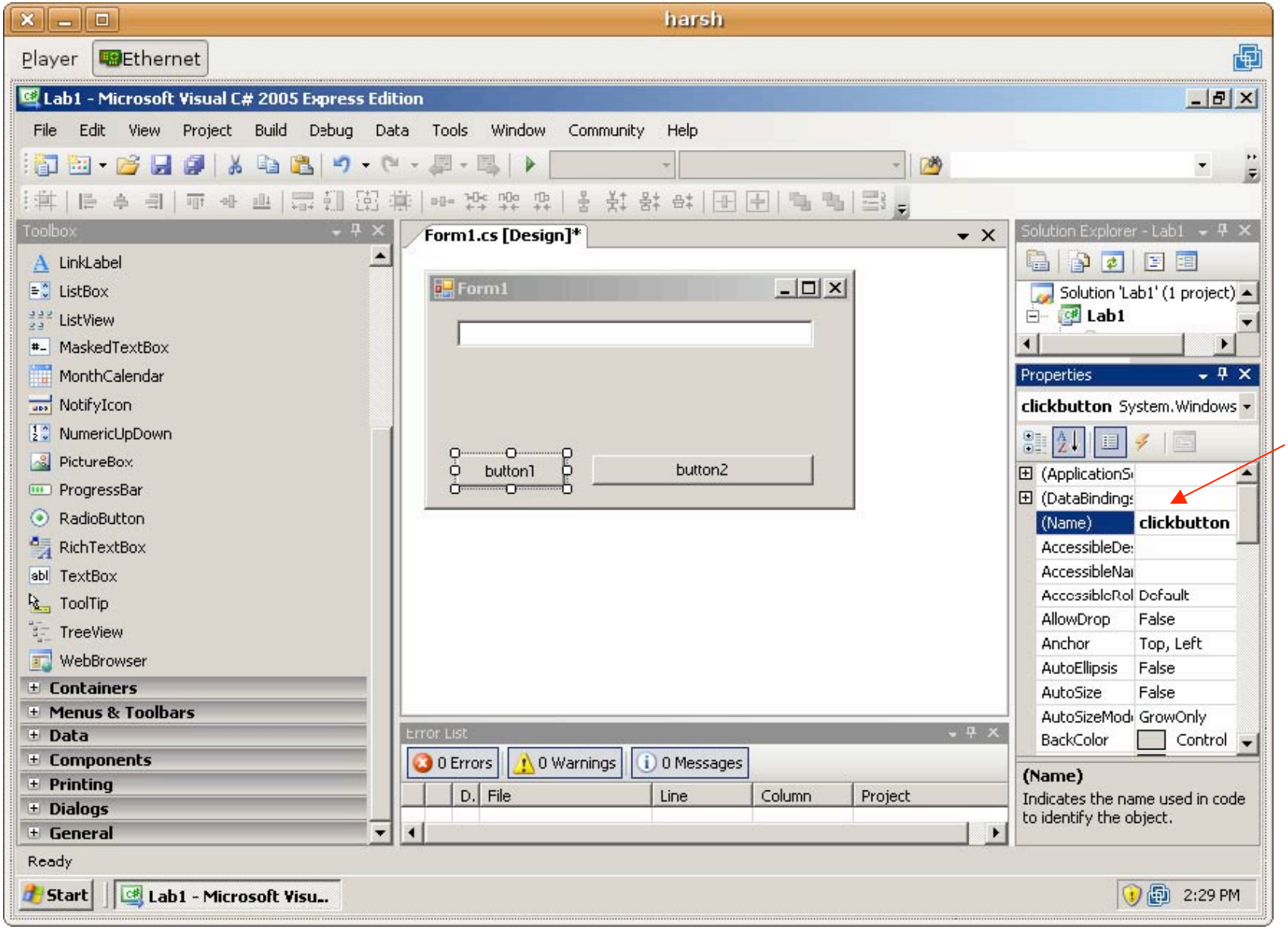


In addition to adding the buttons, add a TextBox. The "TextBox" widget can be found further down under "Common Controls" (depending on the size of the screen you may have to scroll down to find it). Add a TextBox to the form as shown.



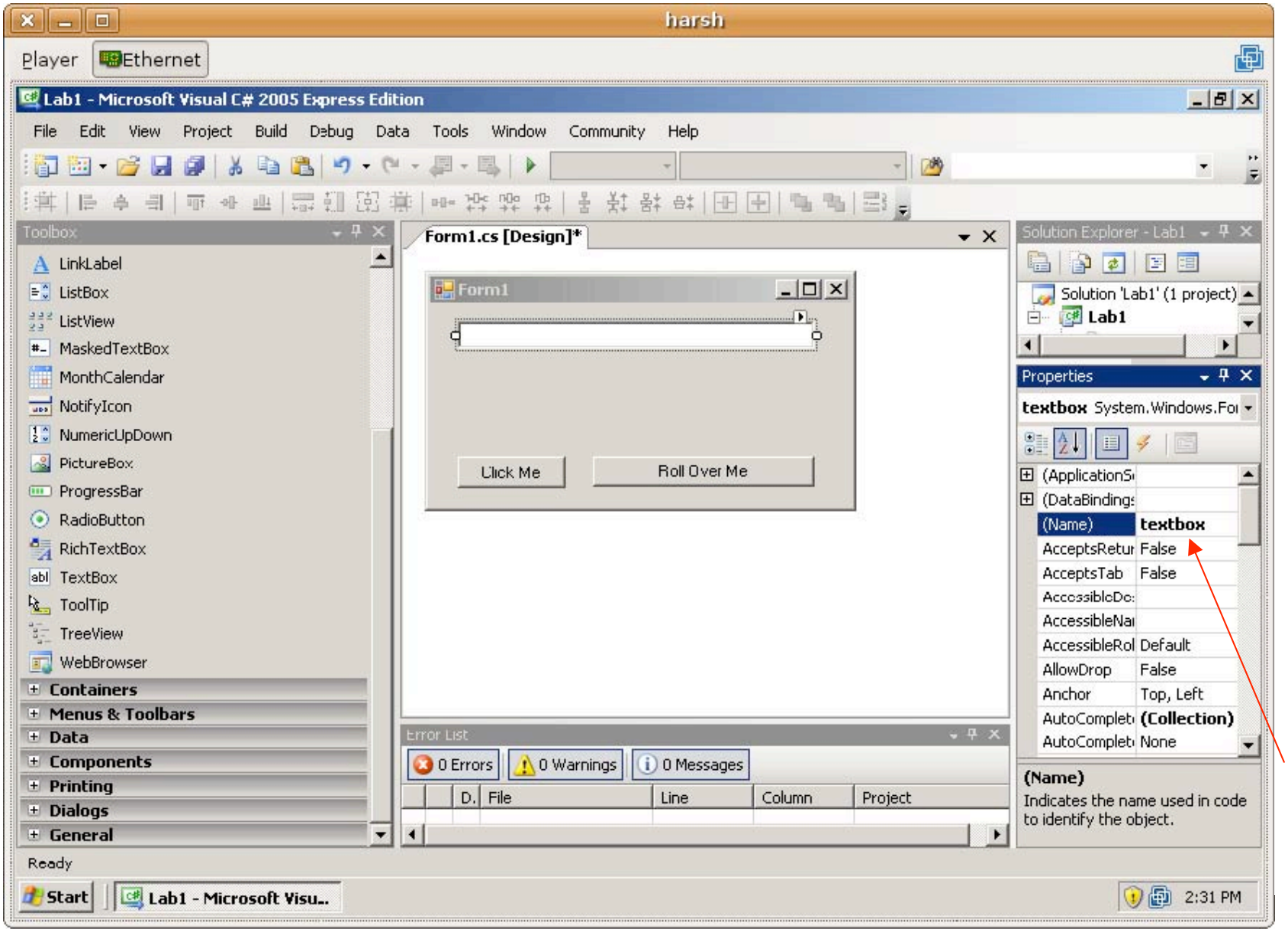

Every widget that is part of the user interface has properties. These properties can be changed during the design of the application and many of them can be changed at runtime by the program itself. One property that cannot be changed at runtime is the "(Name)" property. This is the name by which the widget is internally known. This name is not seen by the user, but will be used by you to refer to the widget in your program code.

The properties of a widget can be modified using the "Properties" window near the bottom right of the Visual Studio UI (user interface). If the properties window is not shown then you can cause it to be shown by selecting the "Properties Window" item on the view menu. There are usually a lot of properties so you'll have to scroll to find them. It may help to resize the properties window to be larger by dragging its top edge.




When modifying properties, the widget that is currently selected in the form layout editor (the one with the border drawn around it) is the widget that you're modifying the properties of.

Select the first button that we created. In the properties list scroll to the top to find the (Name) property. Change the button to be named "clickbutton".




Scroll further down the properties list to the "Text" property. This is the text that is displayed on the widget. Set this to read "Click Me". Notice that the button's appearance in the user interface has now changed.

Experiment with some other properties like font size and colour. Get a feel for the user interface of the form designer.

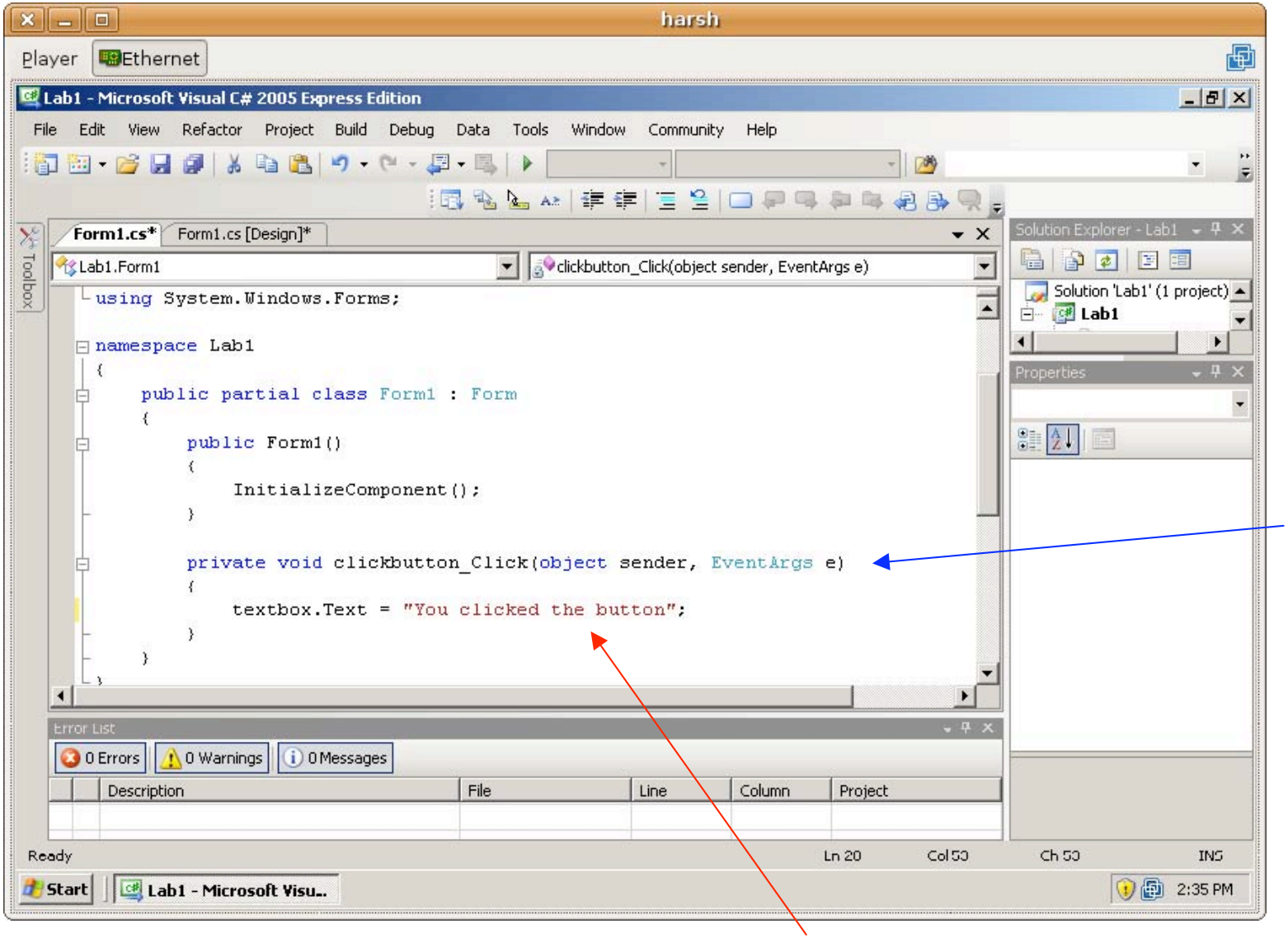
For the second button, change its "(Name)" attribute to "mouseoverbutton" and its "Text" to "Roll The Mouse Over" or something similar.

Change the name of the TextBox to be "textbox" but leave its "Text" empty.

The basis of event-driven GUI programming is that you write small pieces of code that are executed in response to events caused by the user. When an event occurs on a widget you can invoke code (an event handler) to perform a specific task.

In addition to its list of properties, each widget in the user interface has a number of events that it can invoke. These events are invoked in response to user

action. For example, if the user clicks on a button, the "click" event occurs. The list of events available for each widget is available in the properties window. Near the top of the properties window there will be a lightning bolt icon. If you click this icon the properties window will switch from displaying the properties of a widget to the events that that widget is capable of generating.



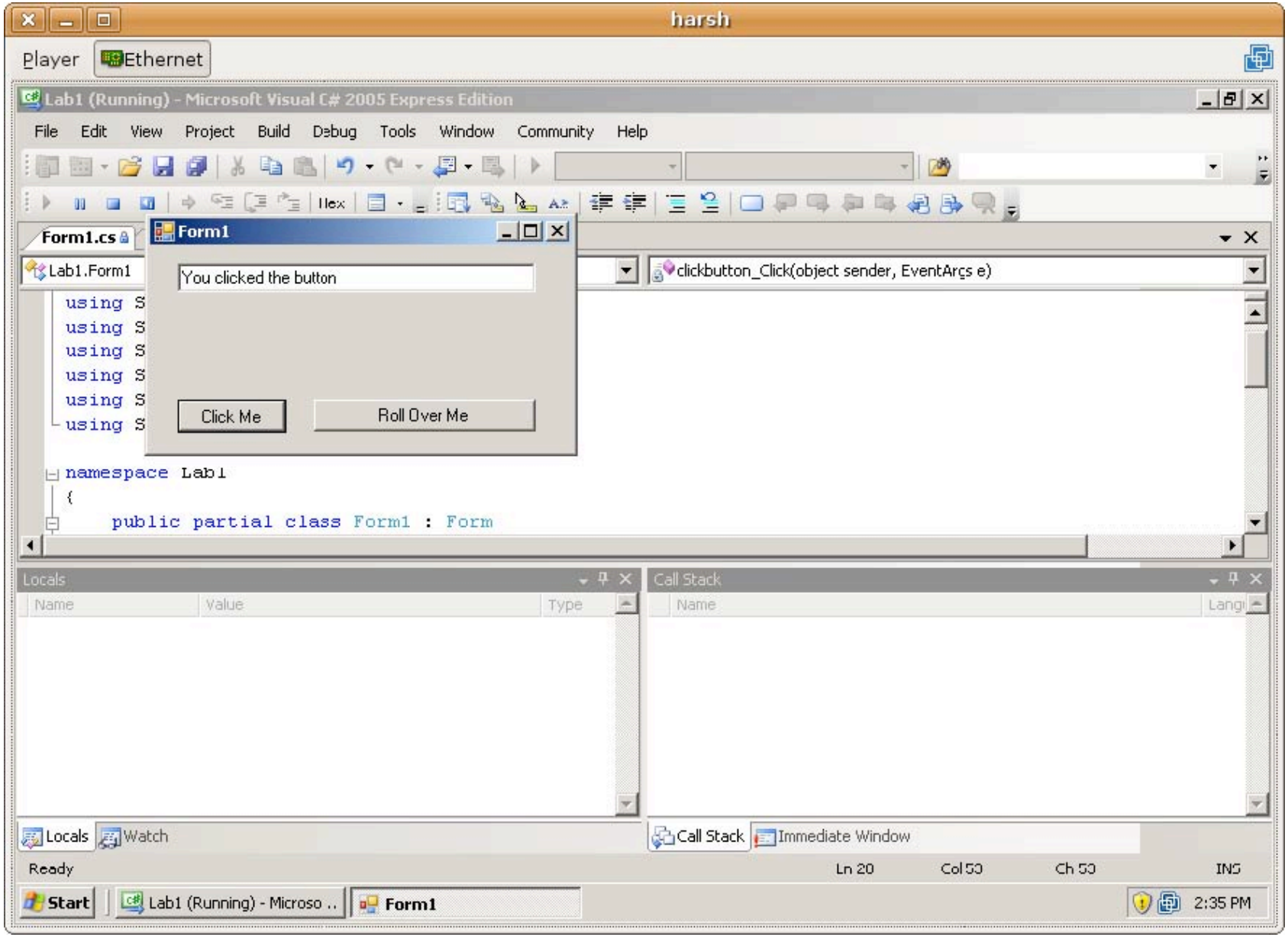

To create an event handler for a widget, first select the widget from the form designer then double click on the event in the event list. For example, to create a "Click" event handler for the button that says "Click Me", first select the button, then from the events list, find the "Click" event and double click on it. At this point, the user interface will change. A new view will open – the code view. This contains the code of your program expressed as a series of event handlers.

A new event handler will have just been created called "clickbutton_Click". The name of the event handler reflects the widget on which the event has occurred (clickbutton) and the name of the event (Click).

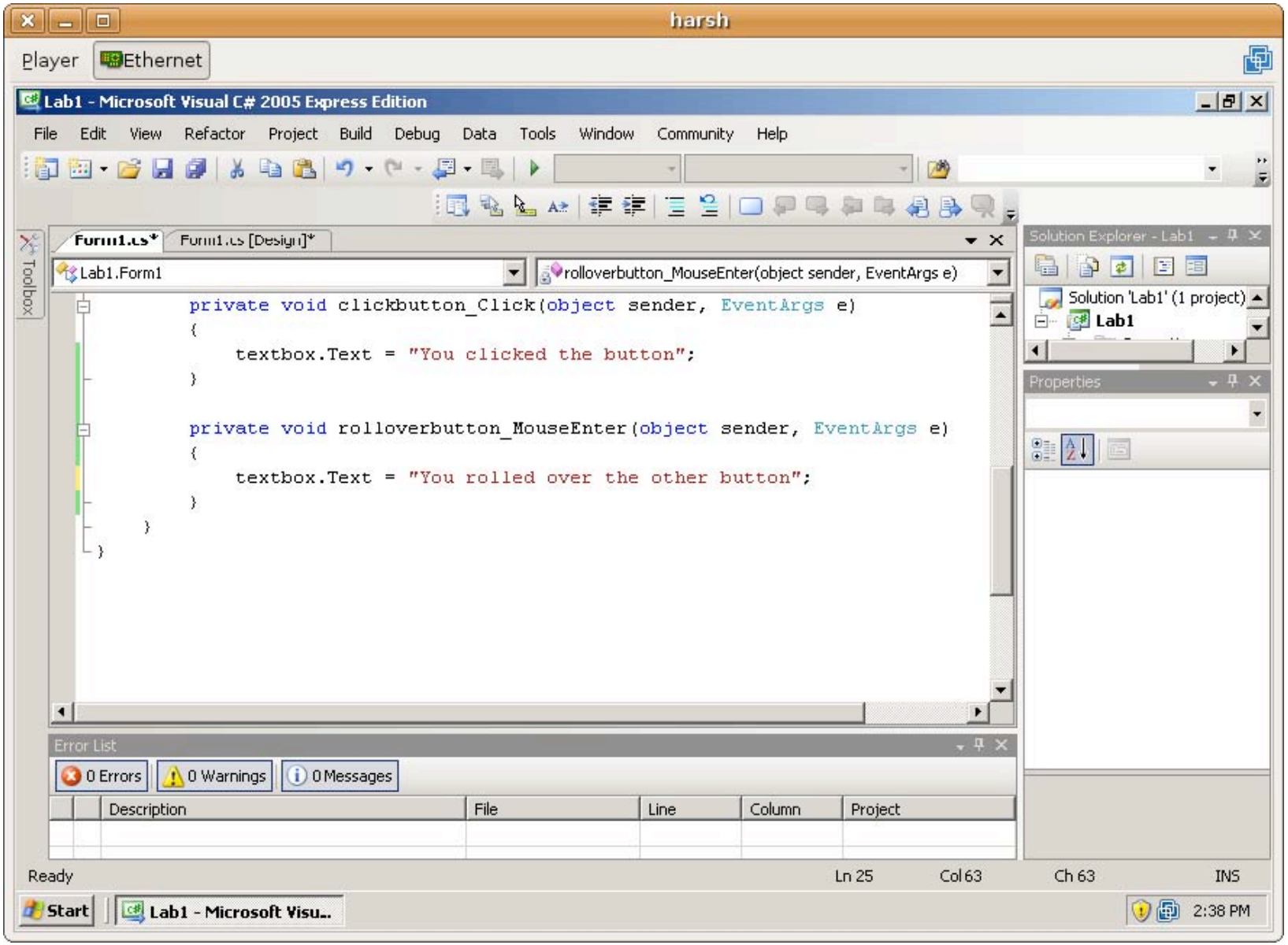
The event handler has a body that begins with { and ends with }. Between these brackets are where you will put statements of program code that will be executed when the given event occurs.

Earlier we said that some properties of widgets can be modified at run time. We're going to do exactly this. Write the indicated line of code inside the body of the clickbutton_Clicked event handler: Remember the semicolon! Now, when the "Click Me" button is clicked on, this code will run.

There is a green "play" button in the UI to test the program. If you click this now then the program should compile and run. If there is an error, ask a TA for help. If everything works out then you should see your application on the screen. You can click the "Click Me" button to run your code to modify the content of the textbox.



Close the program and return to the development environment.




When you created the event handler, Visual Studio automatically switched you to the code view of your program. You can get back to the user interface design view by clicking the "Form1.cs [Design]" tab at the top of the code area. Make sure that the form designer is visible again (i.e. click the "[Design]" tab). Now, similar to the steps above, create an event handler for the second button. This time, use the MouseEnter event (which will occur when the mouse cursor enters the area of the button). Similar to what we did last time add code to this event handler.
You should be able to compile and run the program again to see the result.

Now, return to the form designer. Create a third button with a label of "Reset". When this button is clicked, the textbox should be returned to its original empty state. Write the code required to do this and demonstrate your resulting application to a TA.

Save the project by choosing "Save All" from the file menu.

**NOTES:**
1) **If you choose any of the other "Save" options you will not save all the relevant components of the project. So – it is important that you choose "Save All".**
2) **The projects are stored in folders and each project consists of a number of folders and files. The default location for the project folders is …\My Documents\Visual Studio 2005\Projects**
3) **In order to copy a project (especially to submit it for an assignment) you should zip the folder (top level under …\Projects\) by right-clicking on the folder and selecting "Compressed Folder" from the menu.**

Assignments are submitted through ELM. You may need to submit multiple files for an assignment. Each file has to be uploaded, and only after all files have been uploaded, press the submit button.

For practice:

1. Save your current project in a zip file called Lab1_Test.
2. Copy Lab1_Test.zip to the desktop.
3. Double click LAB1_Test.zip so that the files are extracted, leaving the project folder on the desktop.
4. Double click on the project folder.
5. Double click the file ending in .sln (the solution file). This will run Visual C# and open your project. If it does not work you did something wrong in saving the project, or zipping the project, or extracting the project files. Work out what went wrong and correct it. YOU MUST BE PROFICIENT AT THIS TO SUBMIT YOUR ASSIGNMENTS IN THE FUTURE!

Next lab will end with you submitting the project you create during that lab. After that (sometime in the following week) you will receive feedback from a TA as to whether your submission included all the necessary files. This will be important when you submit your "real" assignments.