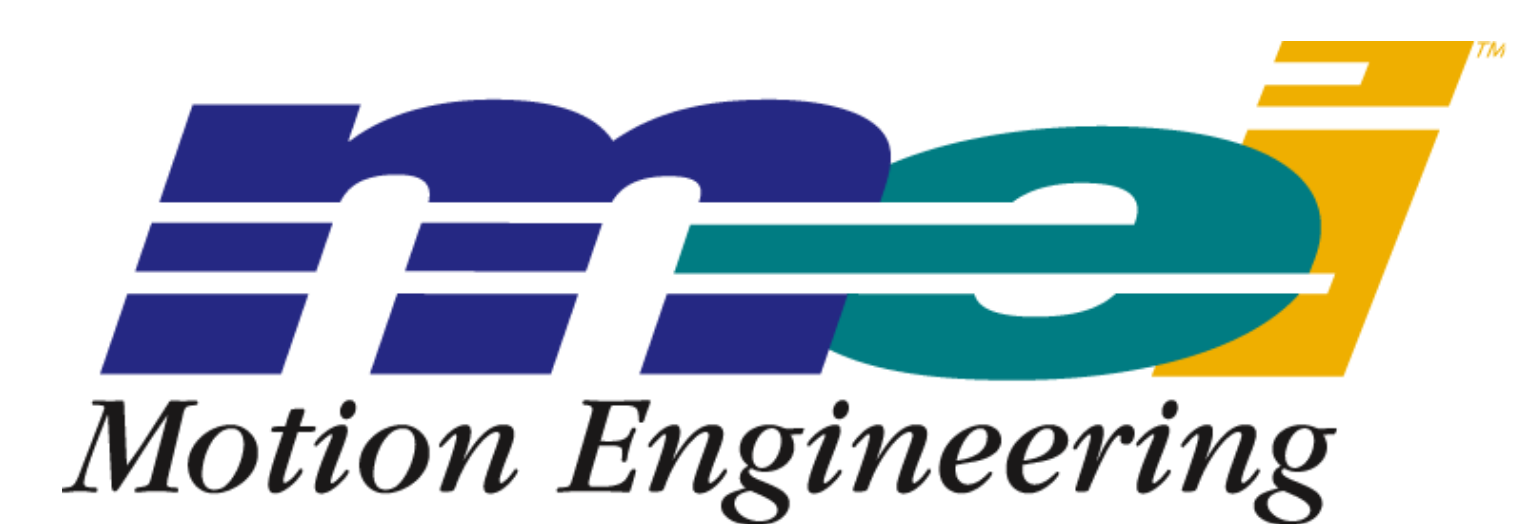




MPX Quickstart Guide for Visual Basic



Copyright Notice

Copyright © 2002 Motion Engineering, Inc.

This document contains proprietary and confidential information of Motion Engineering, Inc., and is protected by federal copyright law. The contents of the document may not be disclosed to third parties, translated, copied, or duplicated in any form, in whole or in part, without the express written permission of Motion Engineering, Inc.

Motion Engineering Inc.
33 South La Patera Lane
Goleta, CA 93117

<http://www.motioneng.com/>

Contents

1	Overview	3
2	Visual Basic Example Application	3
3	Error and Event Handling	8
4	Where to Get Help	11

1. Overview

This guide will take you through the process of adding the MPX control to the Visual Basic development environment and creating a simple application to perform trapezoidal type moves.

This guide assumes the following:

- You have Visual Basic 5.0 or later installed
- You have an MPI compatible controller installed
- You have installed the version of the MPX library compatible with the installed MPI version of software and controller firmware.
- You have either:
 - Hooked up axis 0 to a physical system and tuned axis 0 with Motion Console
 - Or you have set up axis 0 to use stepper motor mode with step-loopback enabled. This will allow motion to be simulated without a physical system being connected to the controller.

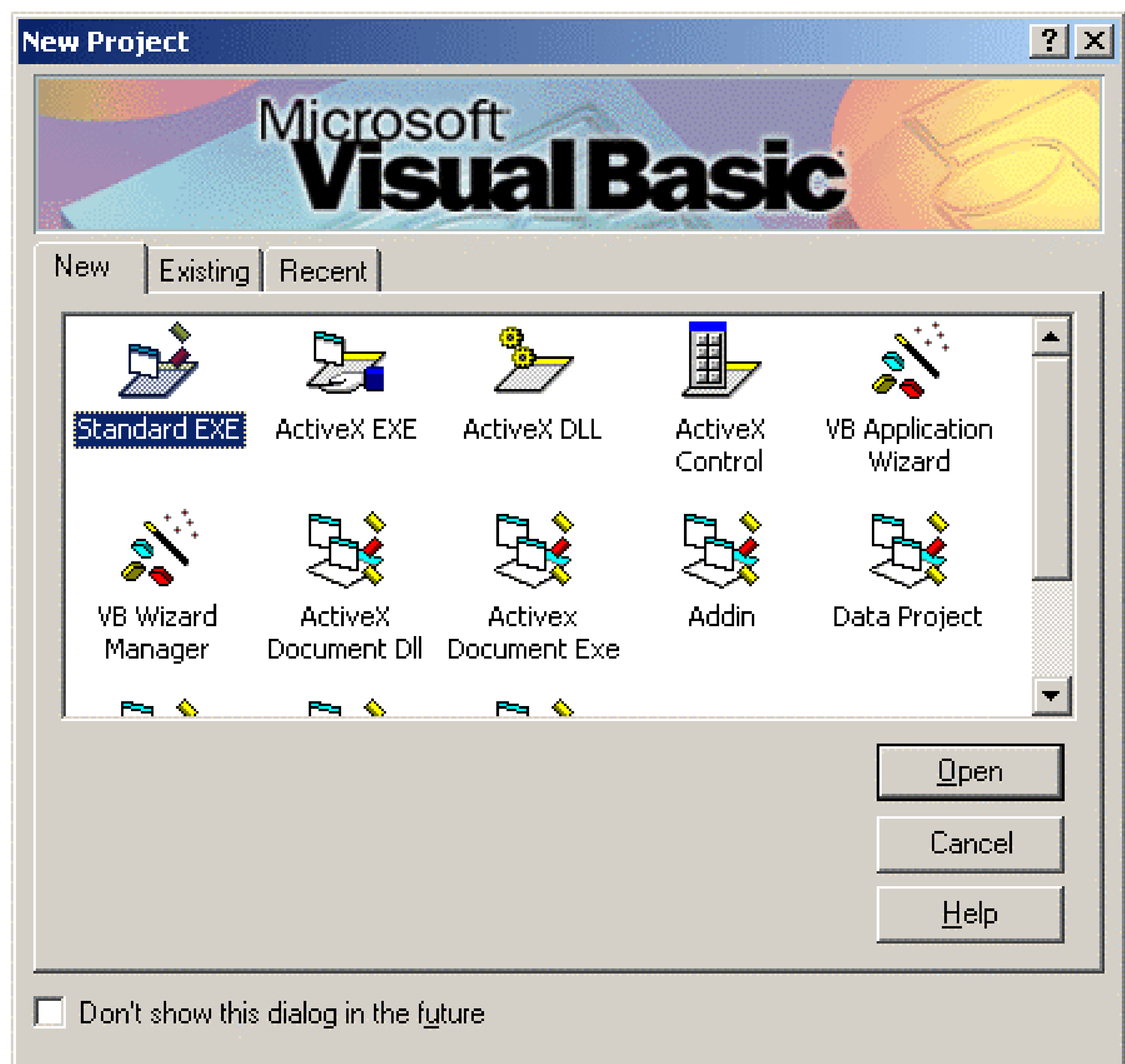
To enable step-loopback mode, please run the Visual Basic script Simulate.vbs, available from the Start Menu:

Start → Programs → Motion Engineering → Simulate.vbs

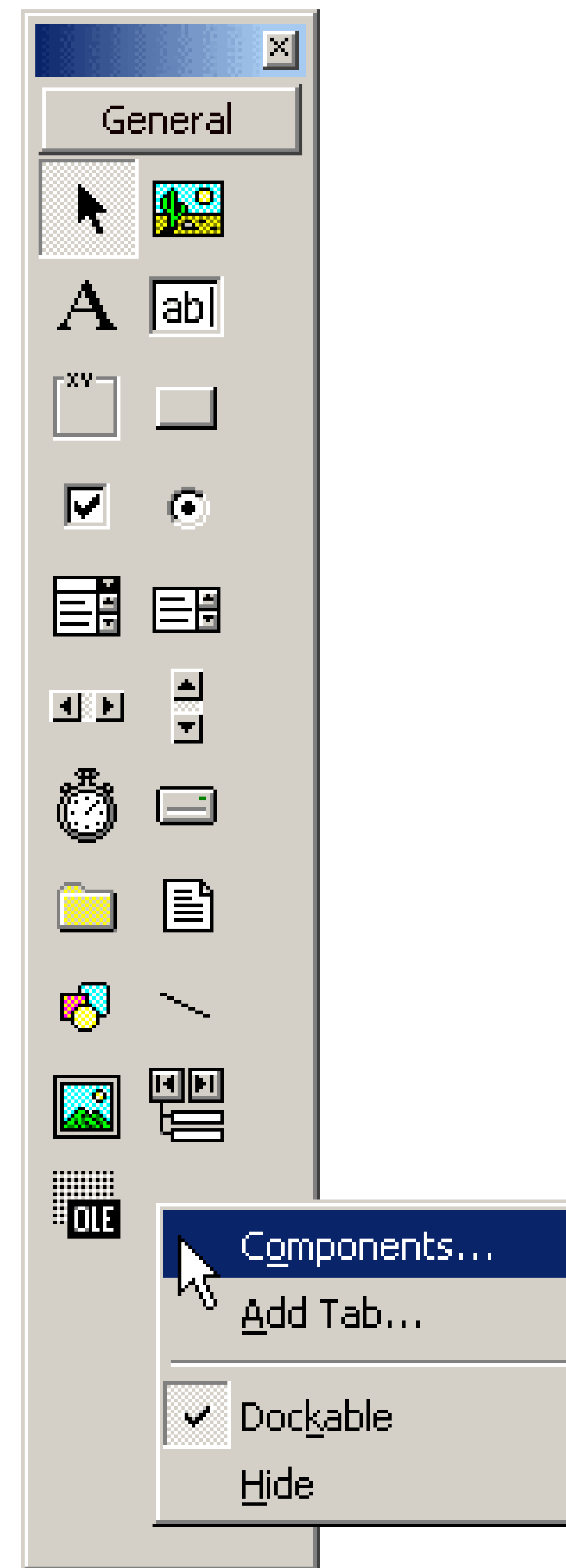
2. Visual Basic Example Application

This example assumes that you have set up axis 0 in Motion Console so that it is already correctly tuned with the amplifier enabled and in an idle state.

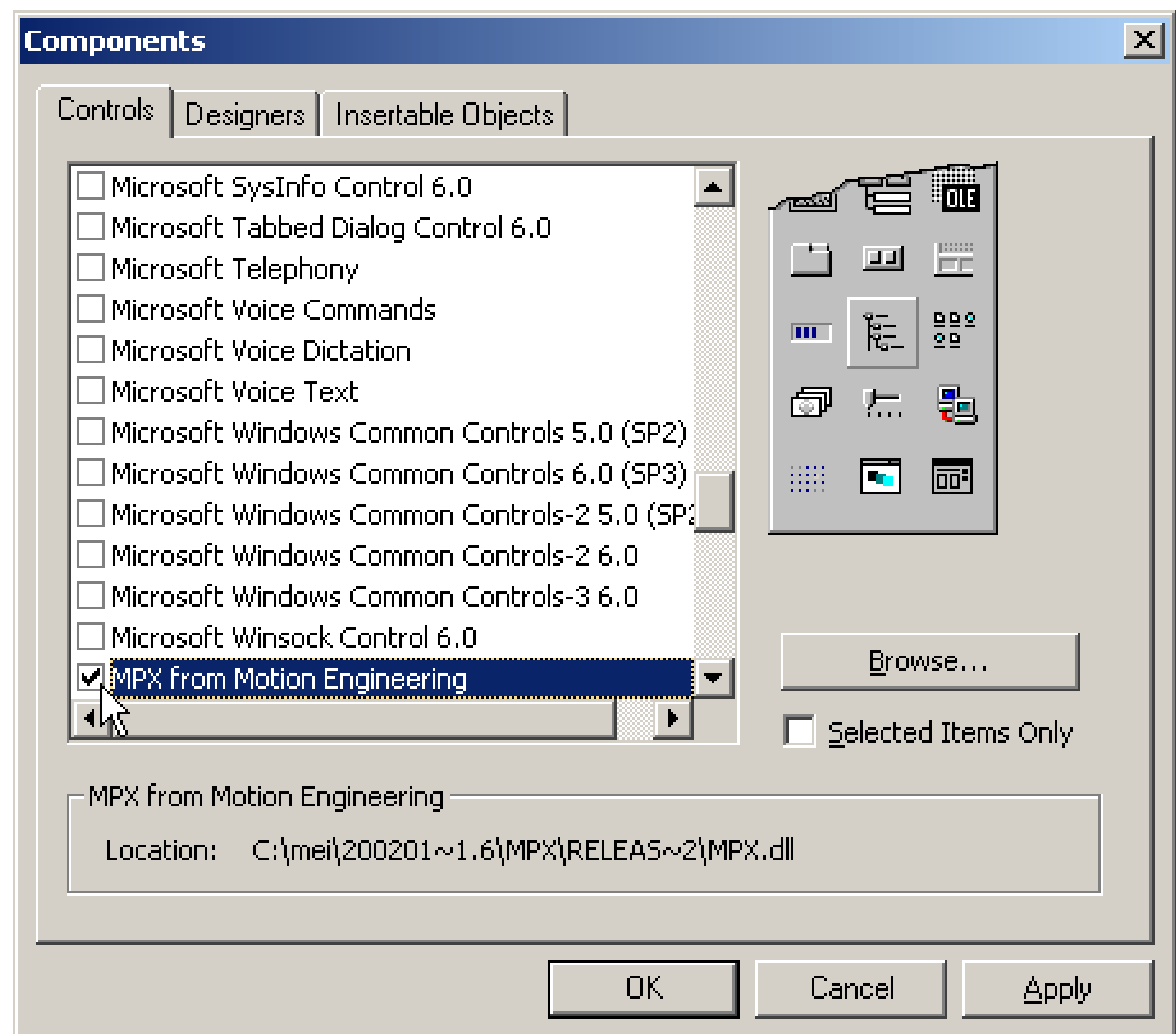
Start Visual Basic. When the **New Project** window appears, select **Standard EXE** and press the **Open** button.



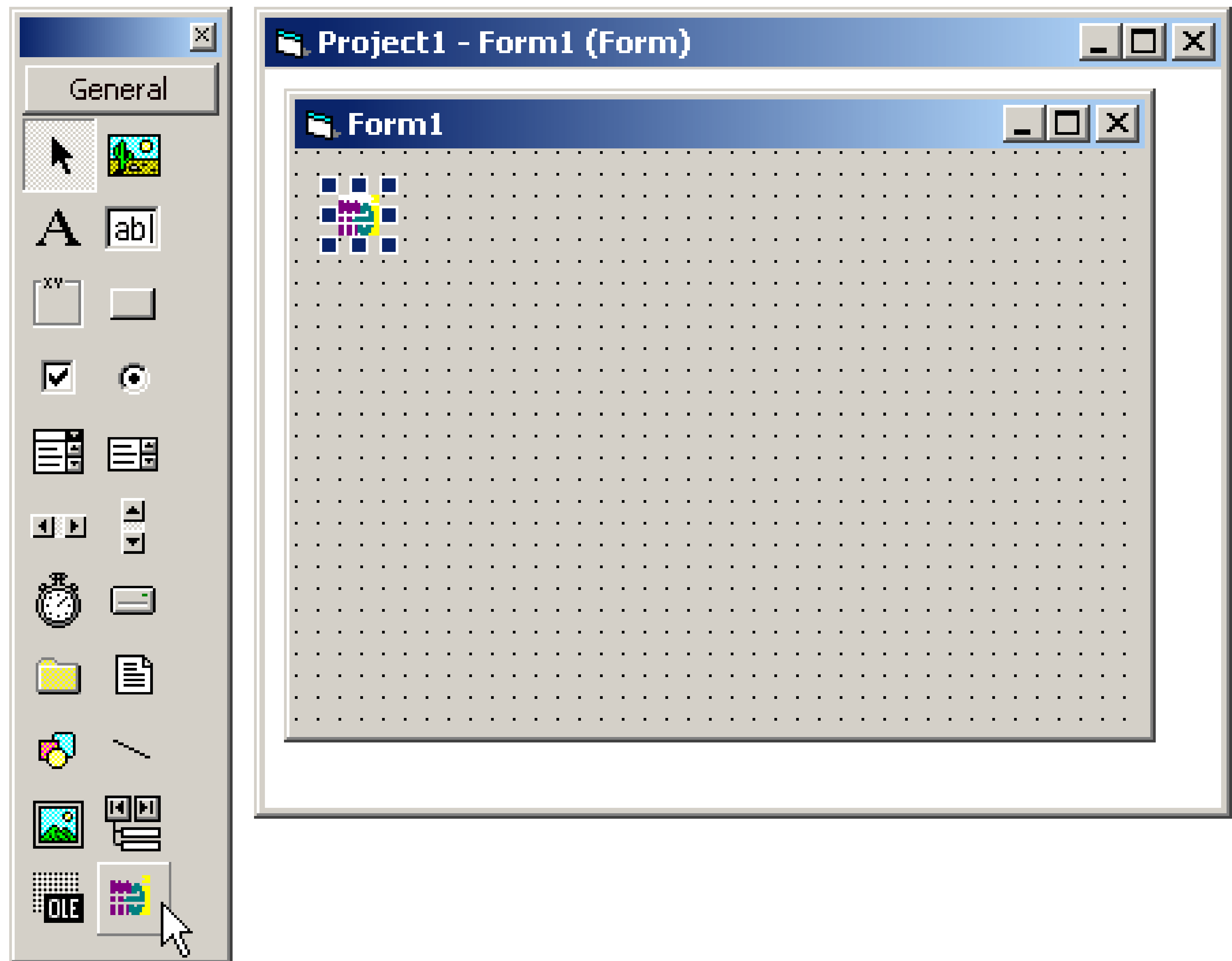
An application with a blank form should appear. Using the mouse, right click within a blank area of the tools window. Select **Components** from the popup menu that appears.



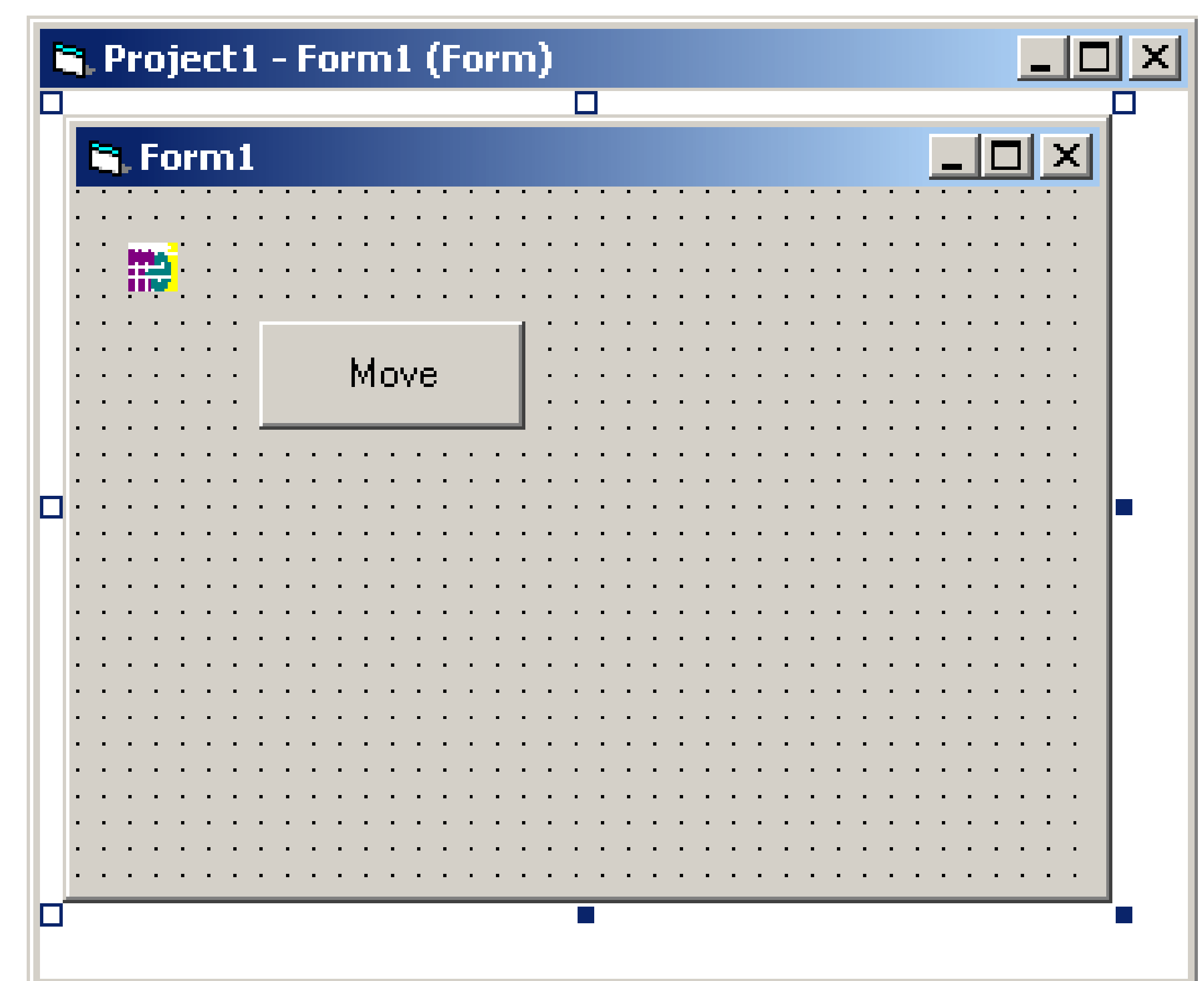
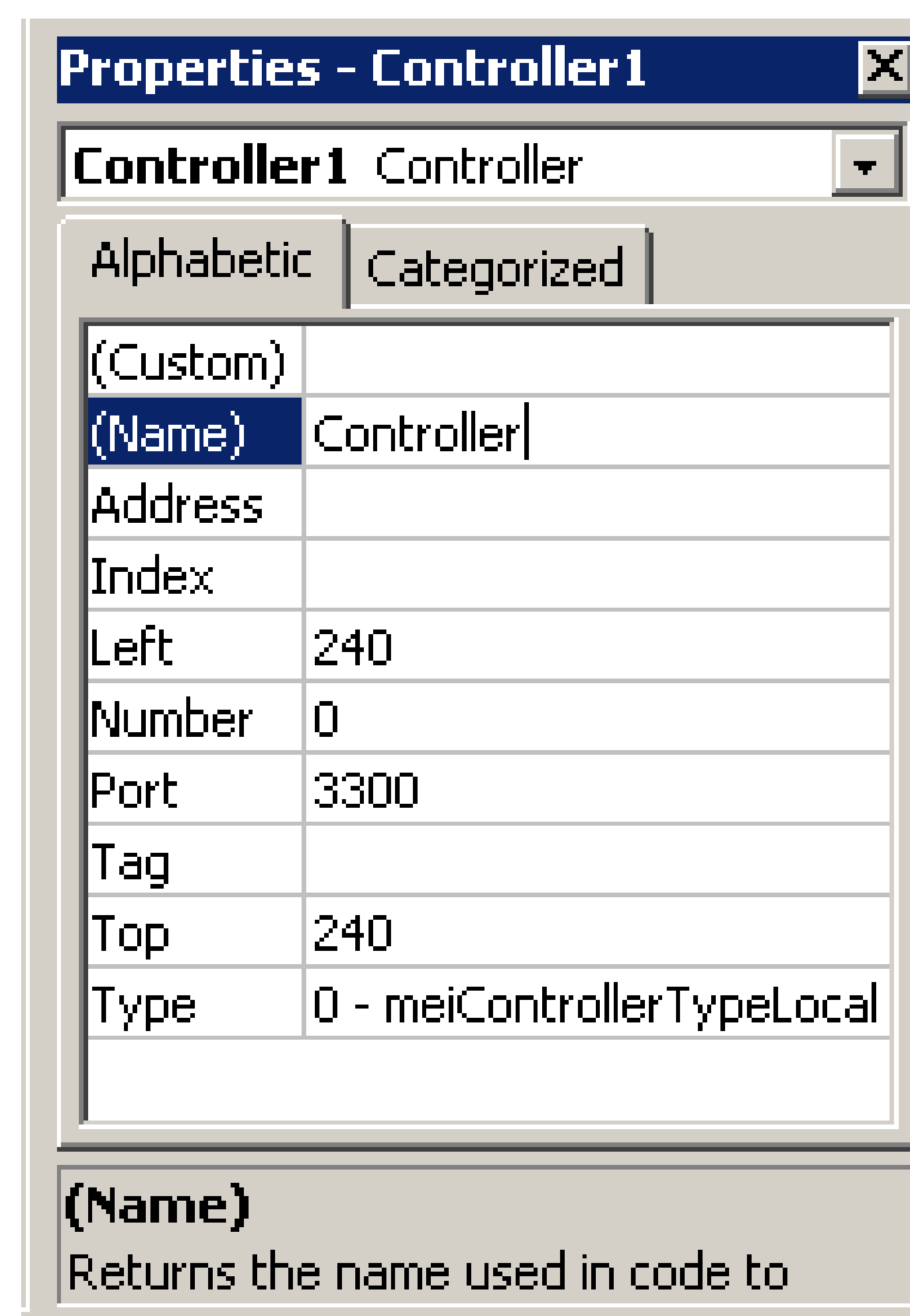
Scroll down until you find **MPX from Motion Engineering**. Select it by clicking the checkbox next to its name and press the **OK** button.



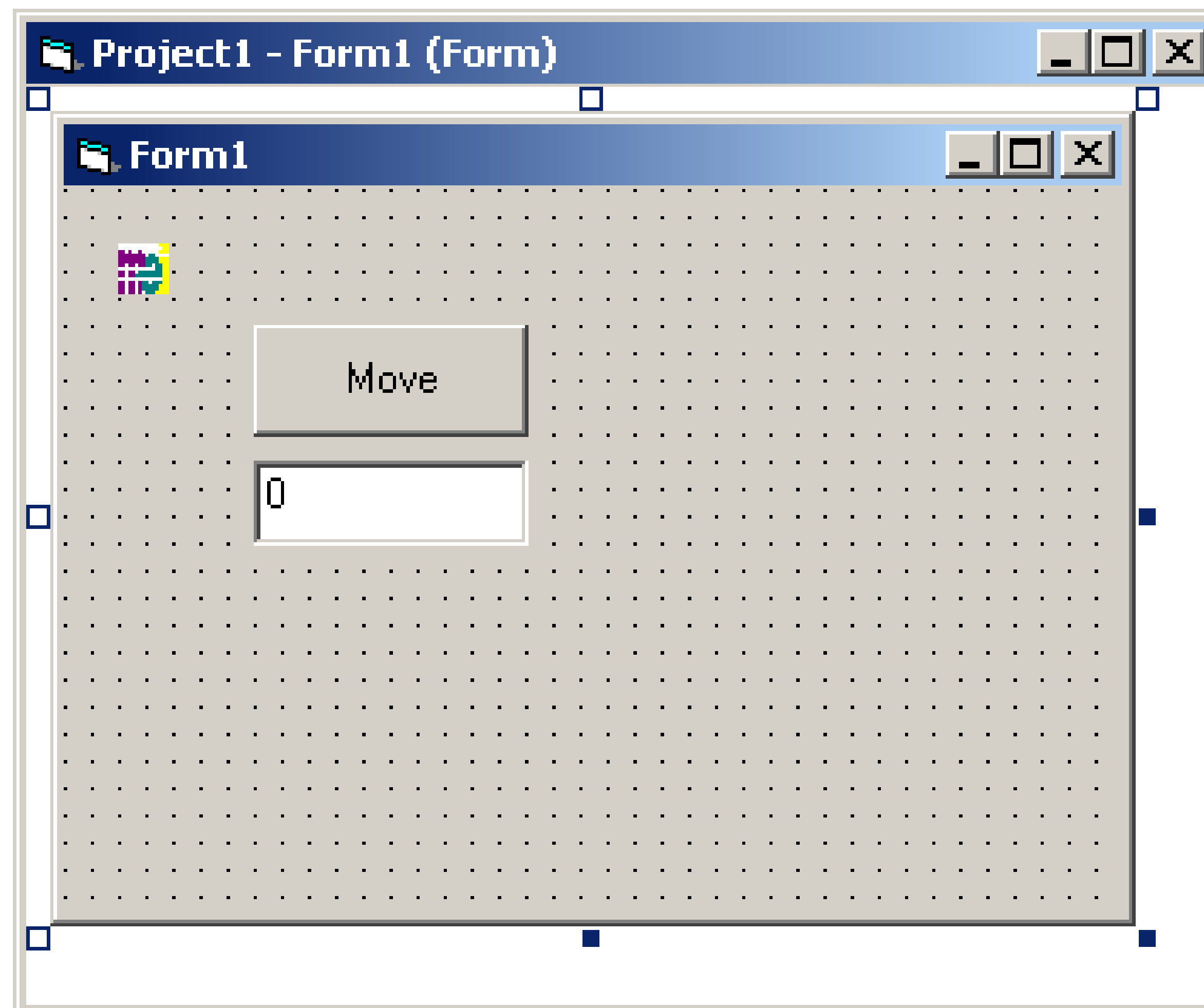
You should now see an MEI icon in the tools window. This is the MPX control. You can add a MPX control to the form by double clicking on the MEI icon.



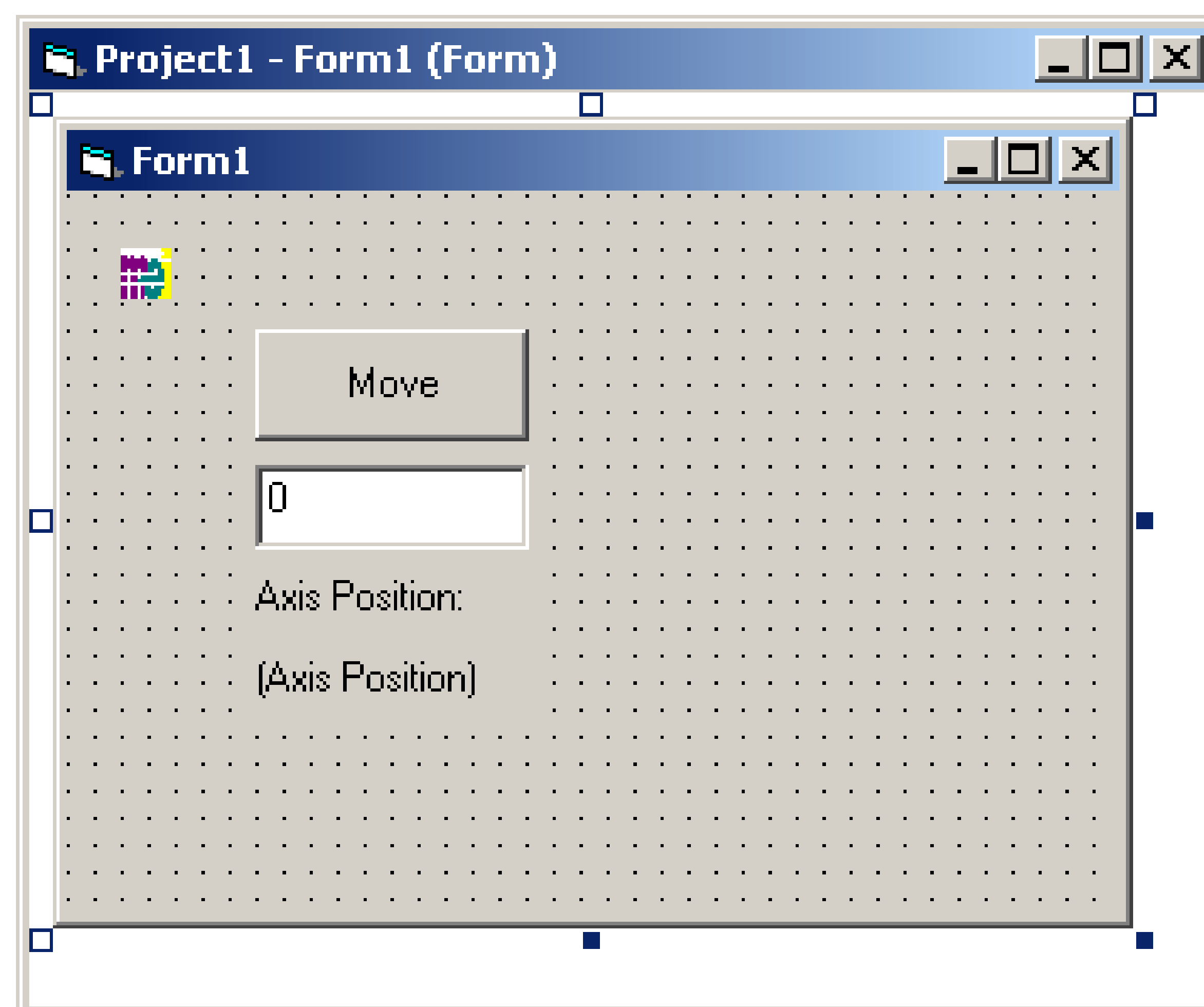
Set the name to “Controller” by changing the “(Name)” property in the **Properties** window. Add a button control to the form by double clicking the button icon in the tools window. Name it “MoveButton” and set its caption to “Move” by changing the “Caption” property in the **Properties** window.



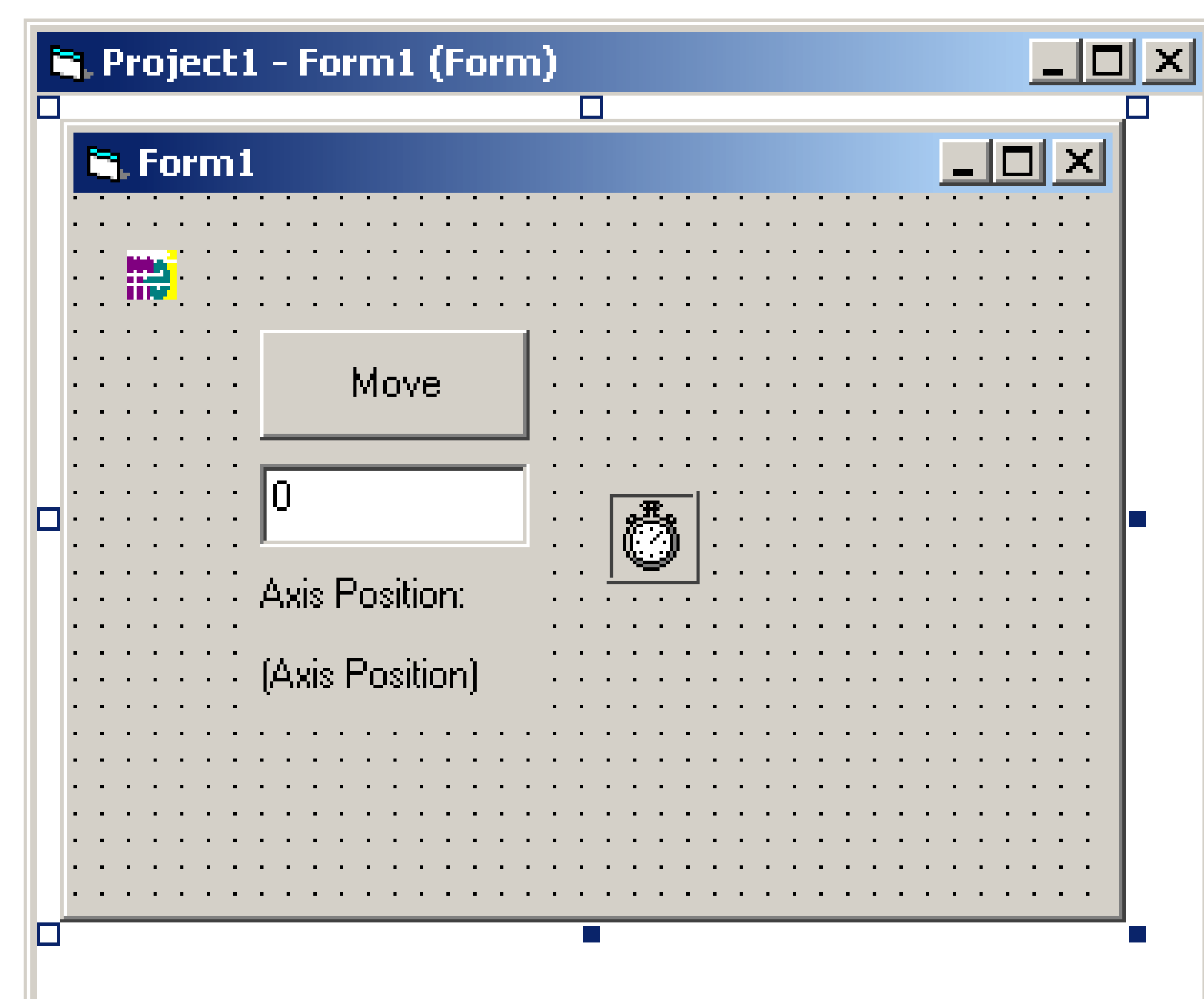
Now add a TextBox control.
Name it “Target” and set its
“Text” attribute to “0”



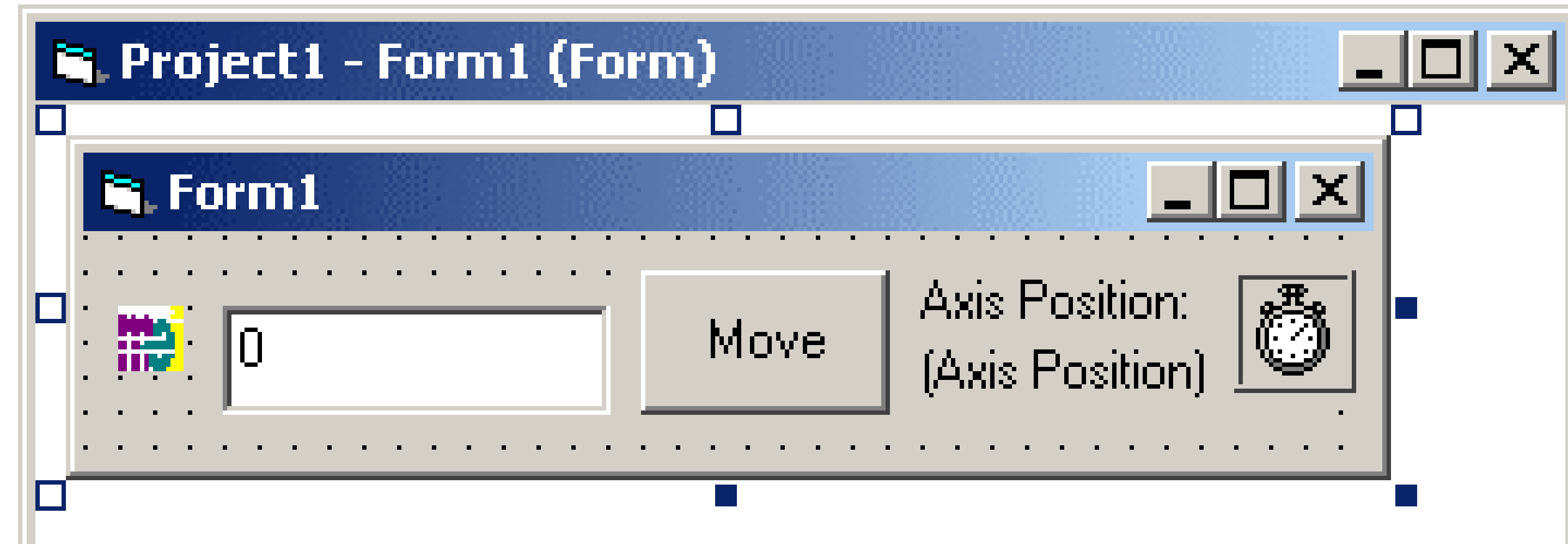
Next add two label controls to
the form. Name one
“PositionLabel” and set its
caption to “Axis Position:”.
Name the other “AxisPosition”
and set its caption to
“(Axis Position)”.



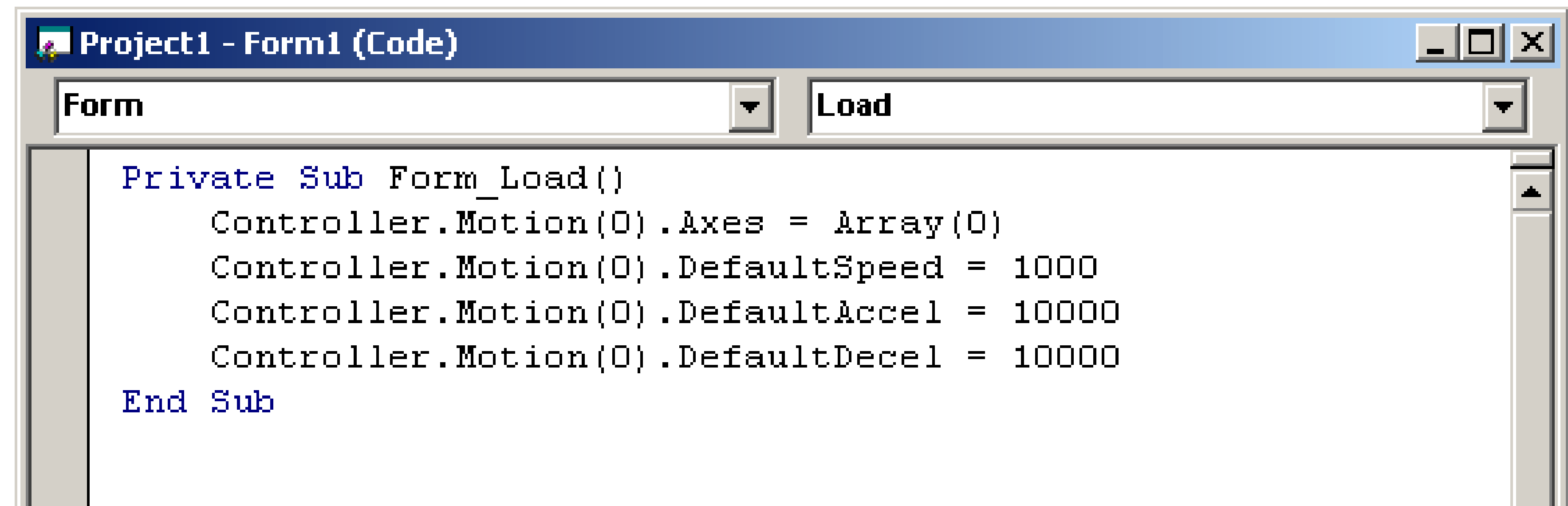
Add a Timer control. Name it
“Update” and change its
“Interval” property to 50.



We are now finished adding the controls we need. Before we go any further, let us arrange the controls and change the size of the form to be more visually appealing.

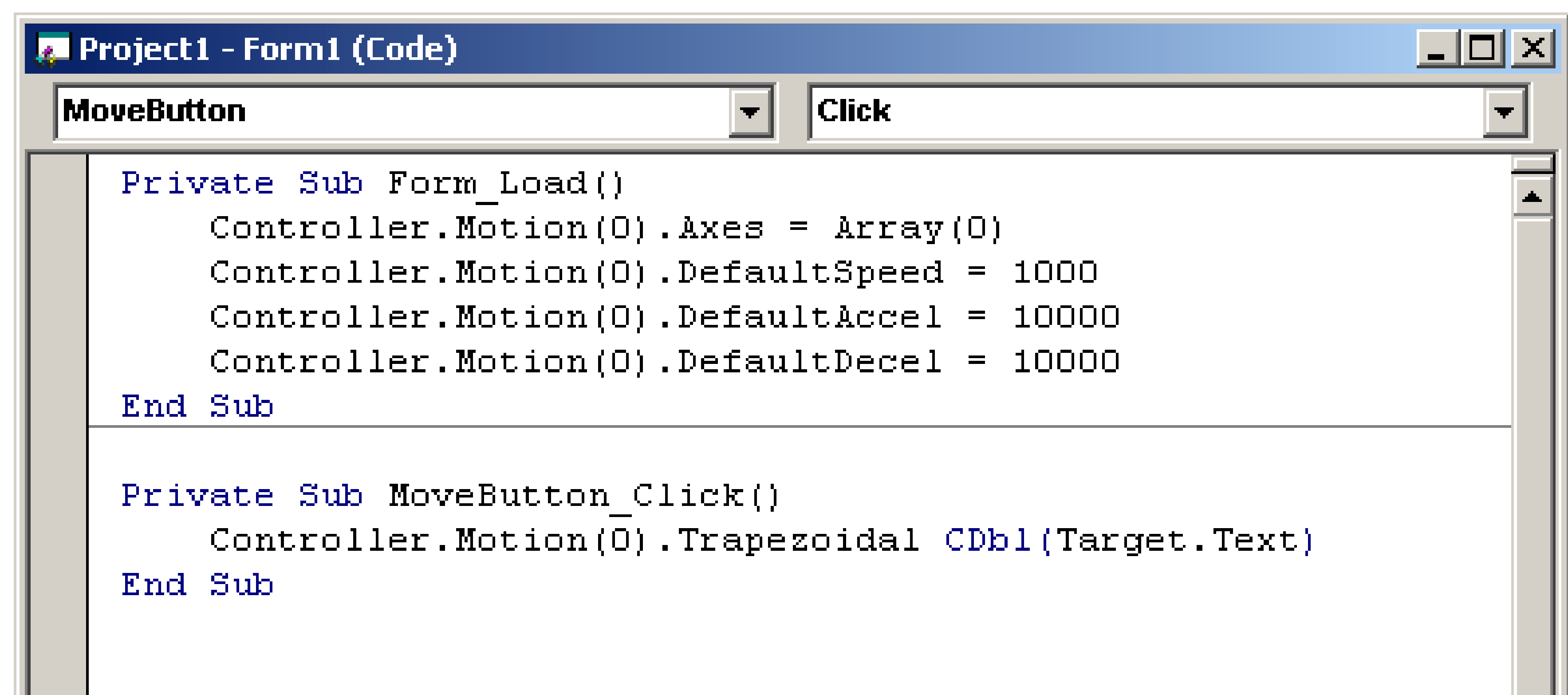


We are now ready to add the code for the application. Double click on a blank area of the form. A code page for Form1 should appear with the outline of the subroutine Form_Load() written. Add in the code shown to the right.

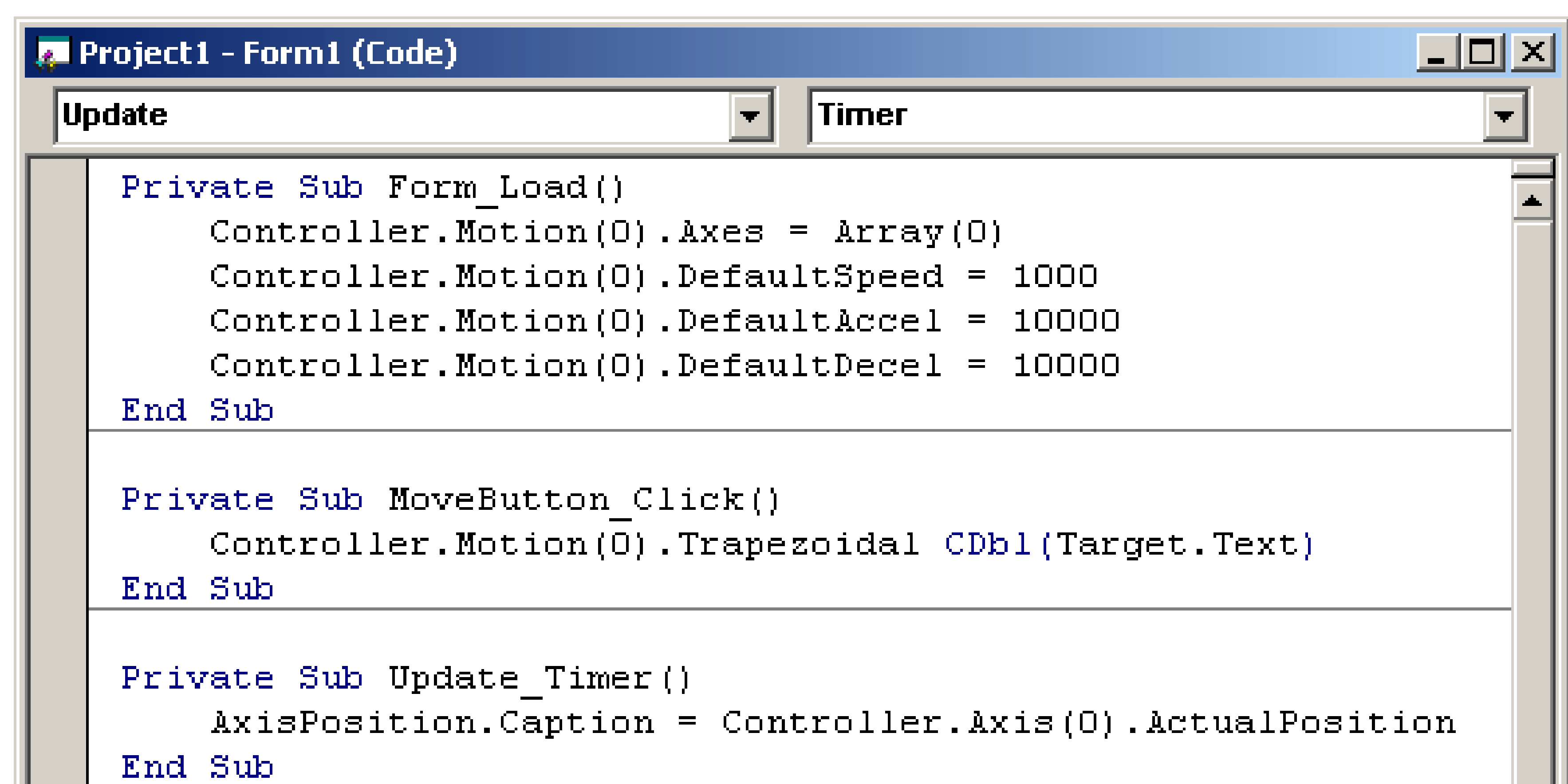


This code tells Motion object 0 to control Axis 0 and sets the default motion parameters for Motion object 0.

Go back to the form layout window and double click on the Move button. Again, add in the code shown to the right.

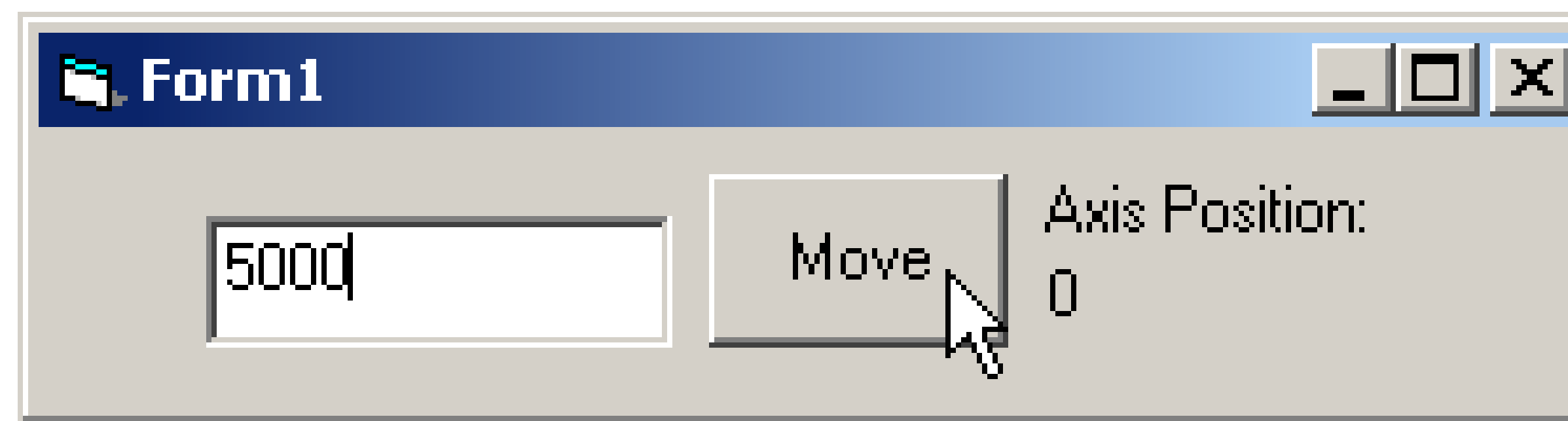


Double click on the Timer control so we can add code to update the display of the axis position.



We are now ready to run our application. Press the **F5** key to start the application.

Enter 5000 in the text box and press the Move button. The axis will move to position 5000 and the current axis position will be displayed.



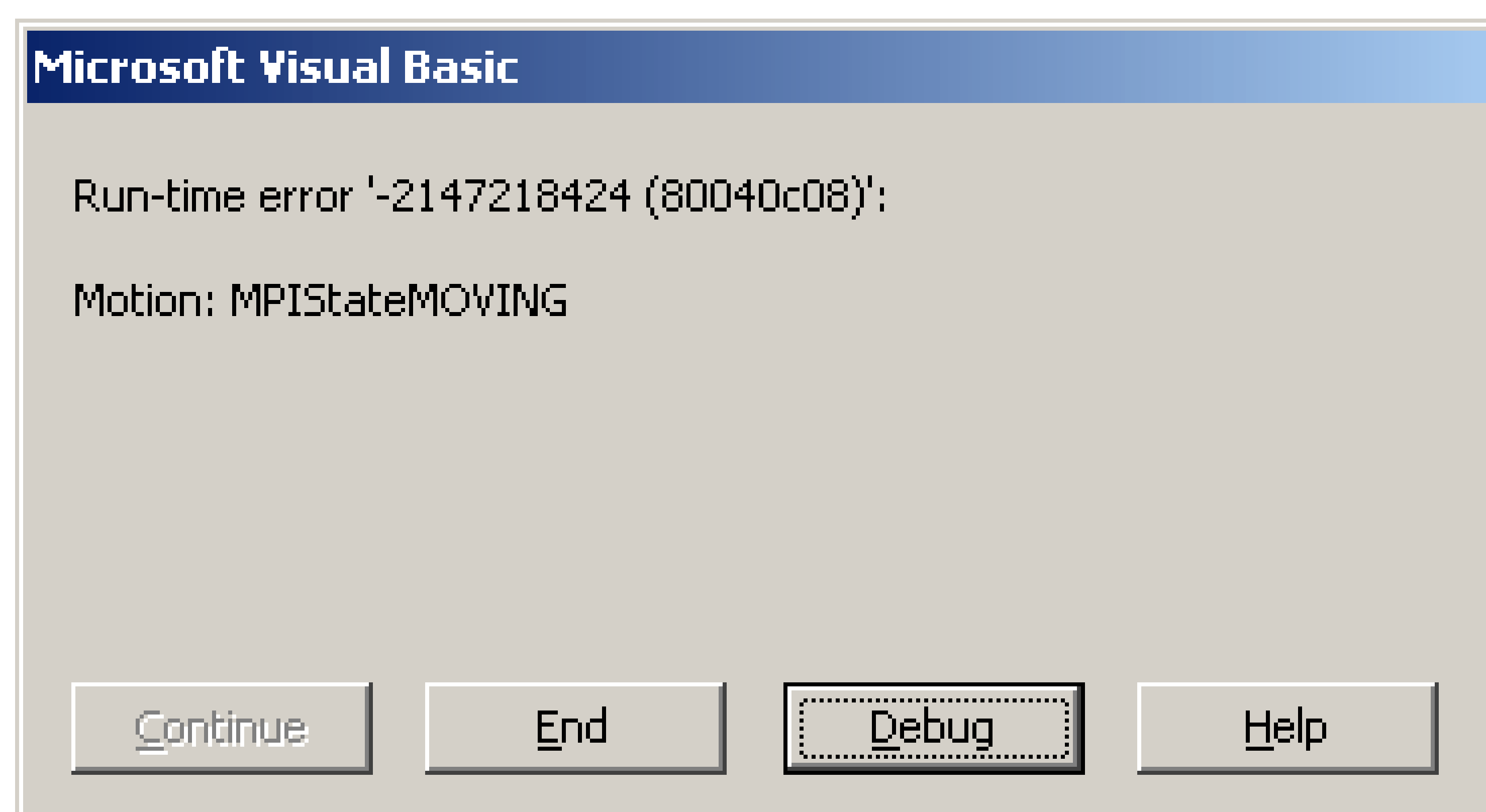
Congratulations, you have completed your first application with the MPX control. Experiment with this application by entering different positions into the text box and pressing the Move button.

If you want to learn a little bit about error handling and events, please continue on to the next section.

3. Error and Event Handling

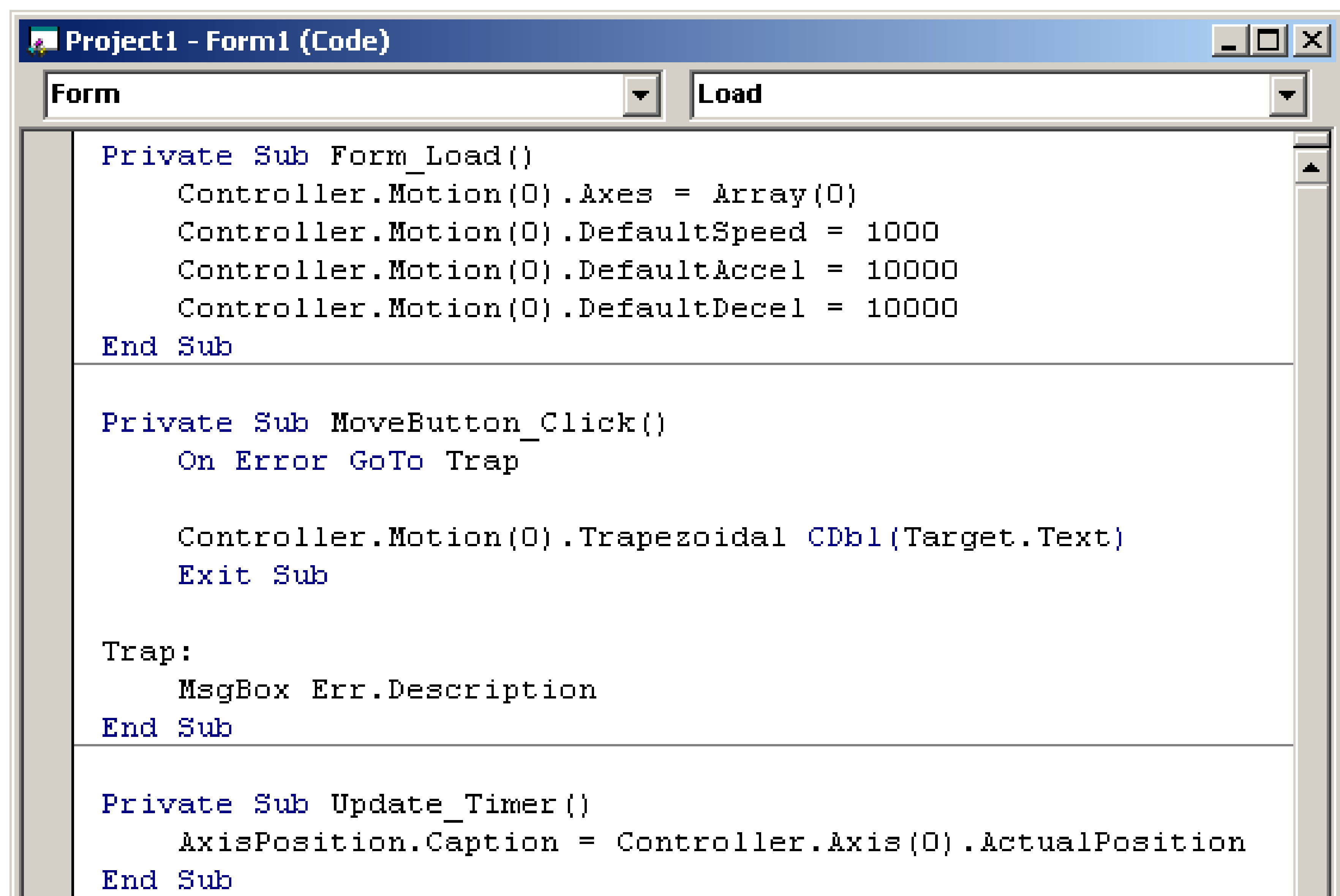
Move the axis back to a position of 5000. Now enter 0 in the text box and click on the Move button twice. The second click will command another motion while the first motion is still being executed. This will cause the ActiveX control to raise a run-time error.

You will notice that a dialog appears announcing the error and from which the application cannot recover. Click on the **End** button to return to the Visual Basic development environment.



In a real application, we would not like an error to terminate our application. One way of solving this problem is to use Visual Basic's error handling mechanism. We are going to modify `MoveButton_Click()` to employ this mechanism.

Go back to the code window for Form1. Add the code shown to the right to `MoveButton_Click()`.



```

Project1 - Form1 (Code)
Form Load
Private Sub Form_Load()
    Controller.Motion(0).Axes = Array(0)
    Controller.Motion(0).DefaultSpeed = 1000
    Controller.Motion(0).DefaultAccel = 10000
    Controller.Motion(0).DefaultDecel = 10000
End Sub

Private Sub MoveButton_Click()
    On Error GoTo Trap

    Controller.Motion(0).Trapezoidal CDbl(Target.Text)
    Exit Sub

Trap:
    MsgBox Err.Description
End Sub

Private Sub Update_Timer()
    AxisPosition.Caption = Controller.Axis(0).ActualPosition
End Sub

```

Now execute the application again by pressing **F5**. Move the axis to position 5000 and enter 0 in the text box. Now click on the Move button twice.

You will notice that message box appears instead. And best of all, when you press the **OK** button, the application will still be running.

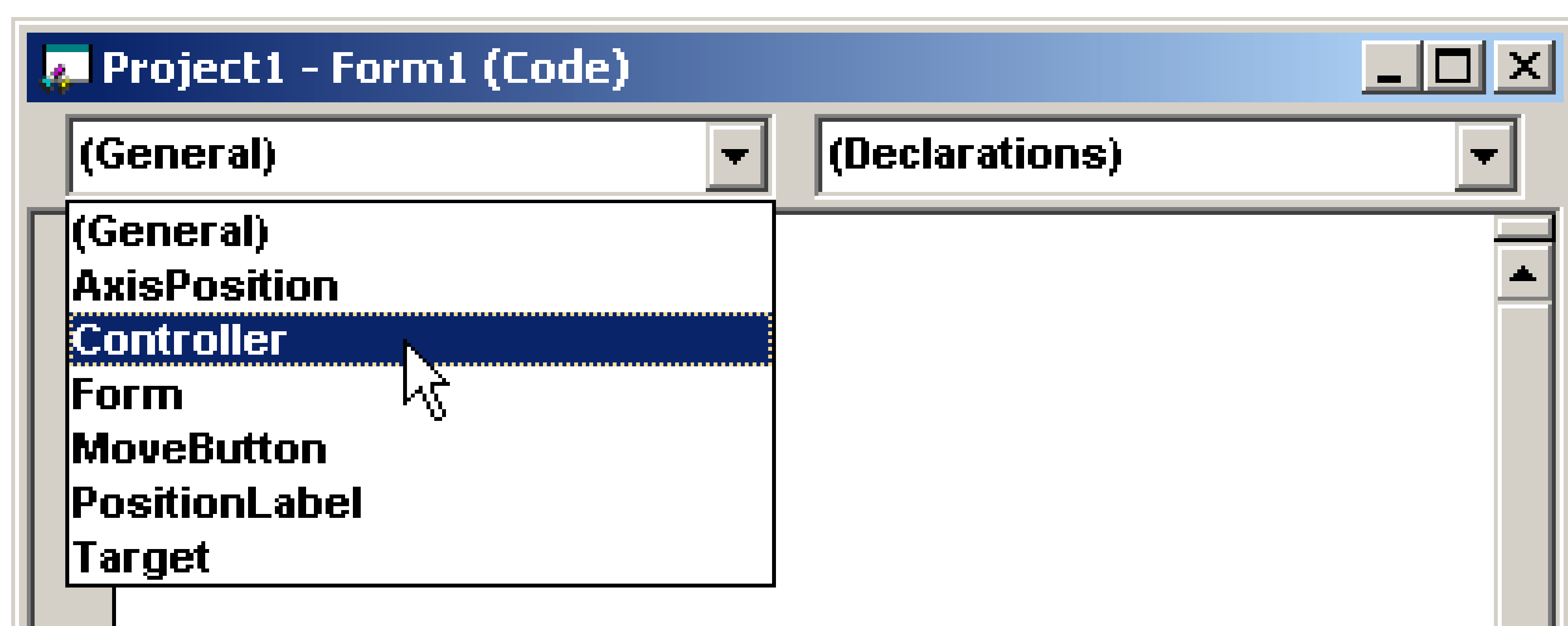


Being able to trap run-time errors is good, but it would be better if we could prevent errors. We could prevent this error if we disabled the **Move** button while a motion was in progress. This would involve:

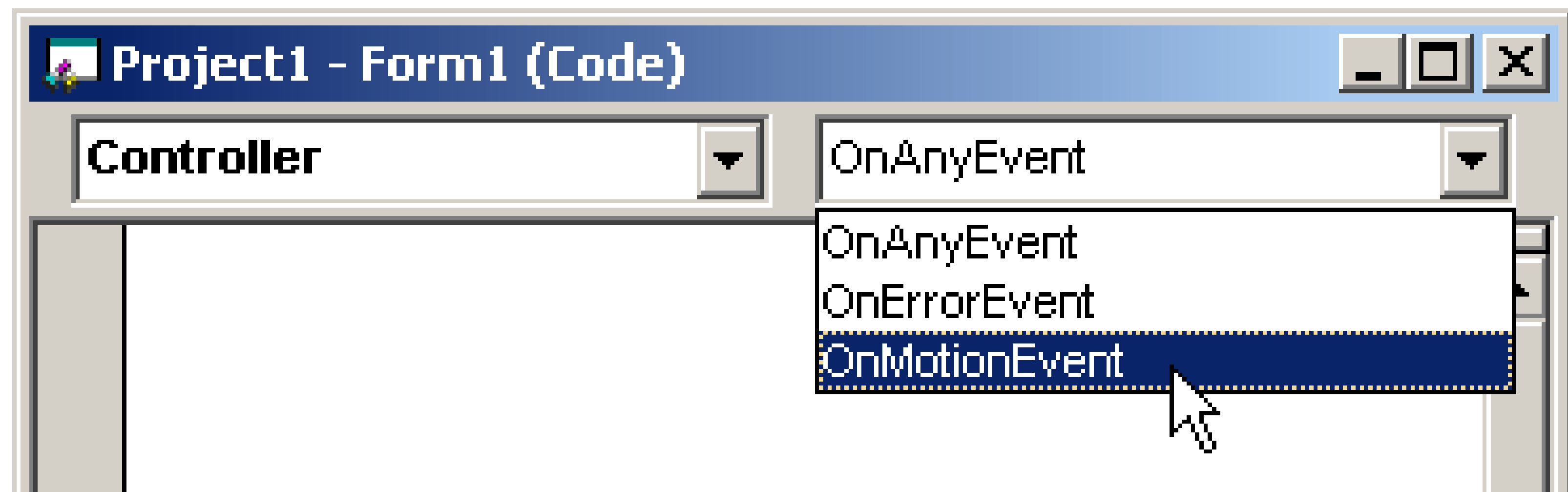
- Disabling the Move button after the user clicked it.
- Enabling the Move button once the motion was complete.

The first requirement seems easy to implement, but we can only implement the second if we can have the controller notify us that the motion is complete. Well, the MPX control allows this to happen through ActiveX events.

To define an ActiveX event callback function, we select the object creating the event from the left drop down box at the top of the code window—In this case we will select the **Controller** object.



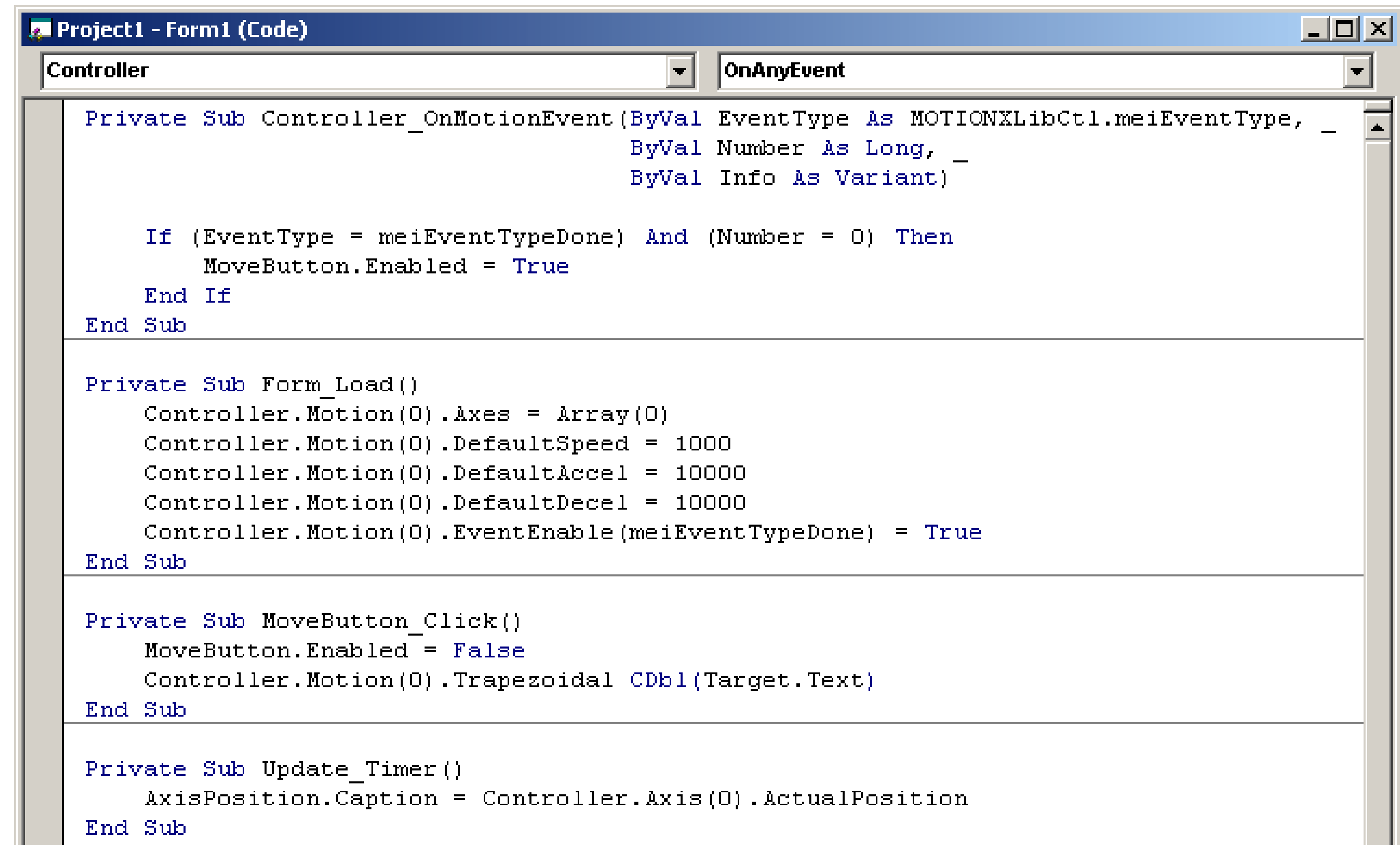
Once we select the object, we need to select the event handler from the right drop down box at the top of the code window. We are interested in the Motion event *Done*, so we will select the handler **OnMotionEvent**.



The mouse will now be placed in a subroutine called **Controller_OnMotionEvent**.

We need to add code to this subroutine and to the **Form_Load** and **MoveButton_Click** subroutines as well.

Modify the code so that it is in agreement with the code shown to the right.

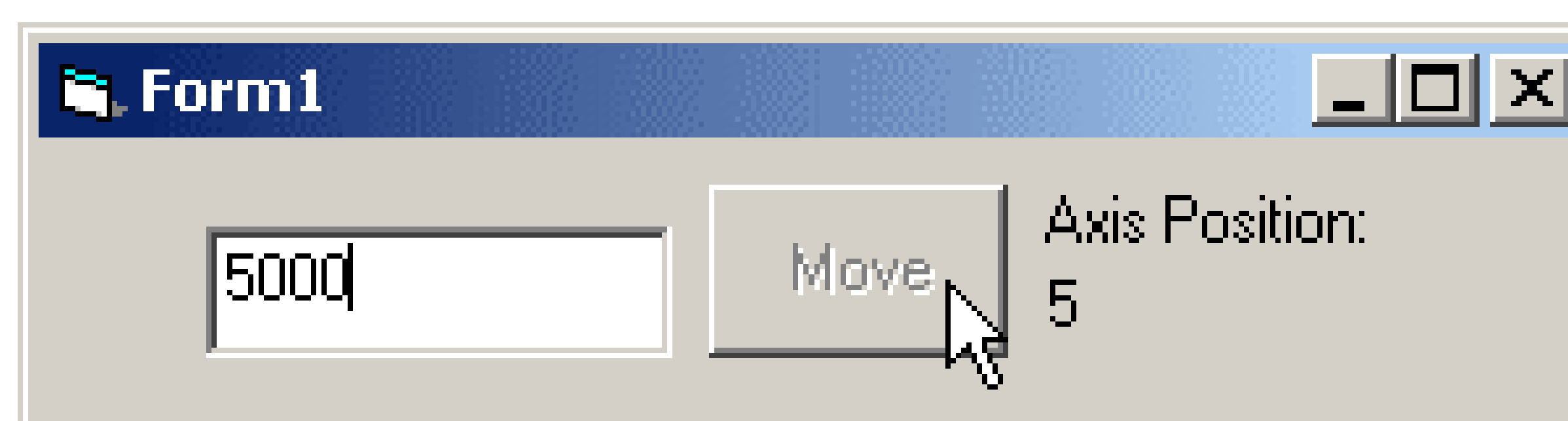


The new code in the subroutine **MoveButton_Click** will disable the **Move** button when a motion is commanded.

The code in the subroutine **Controller_OnMotionEvent** will enable the **Move** button when a *Done* event is received from Motion object 0.

The EventEnable call that was added to the subroutine **Form_Load** enables the reporting of the *Done* event. Without this call, the **Controller_OnMotionEvent** subroutine would never be called and the **Move** button never reenabled.

Now when you run the application and click on the **Move** button, the **Move** button will be disabled until the move is complete.

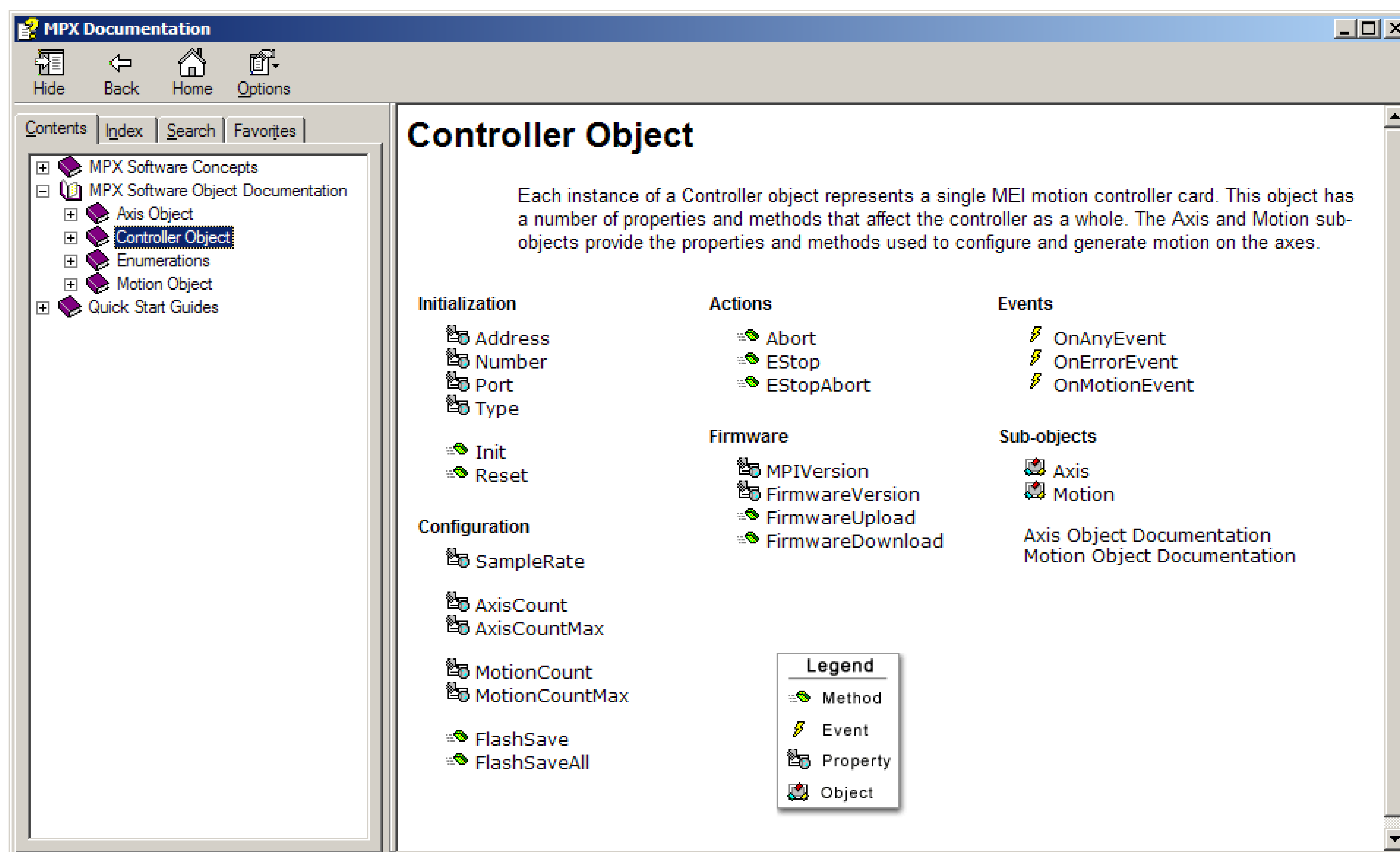


Well done. You have now learned a little bit about handling errors and events. For more sample applications, please see the *ActiveX Visual Basic Sample Applications Guide*.

4. Where to Get Help

A help file is provided with the MPX control. It can be accessed via the Start Menu:

Start → Program Files → Motion Engineering → MPX → MPX Documentation



Also available is our online help system. It can be accessed at <http://support.motioneng.com>

