



Test Automation FX

UI test automation for .NET developers

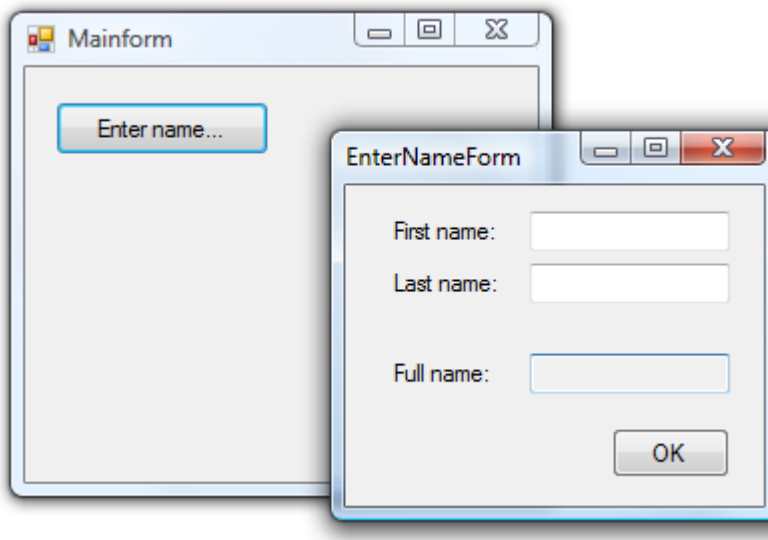
Remap objects

One major problem in User Interface Test Automation is to handle the case when the user interfaces of the tested applications changes. If the test automation tools used heavily mixes UI mapping code and test logic this can be a daunting task.

TAFX makes this easier since it separates the UI mapping code from the test logic. It also has built in support for remapping user interface elements in the UI map designer.

Example

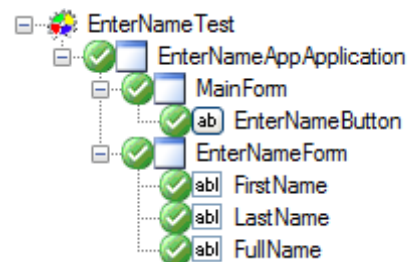
In this example we will take a look at an application for which we have a working UI test. We will then see how we can adopt the tests when the application under test changes. The application we will test looks like this:



It has a Mainform with an **Enter Name...** button that will bring up an EnterNameForm where the user can enter the name information.

We have a test that we have recorded that looks like this:

```
[UITest ()]
public void Test1()
{
    EnterNameButton.Click();
    FirstName.Click(44, 8);
    Keyboard.SendKeys("Tom");
    LastName.Click(31, 15);
    Keyboard.SendKeys("Jones");
    FullName.VerifyProperty("Text", "Tom Jones");
}
```



The test logic and the UIMap structure for the initial version of our EnterName application.

Now, for this example, we receive a new version of the application where the user interface has changed a bit. In the new application the EnterNameForm has been removed and the

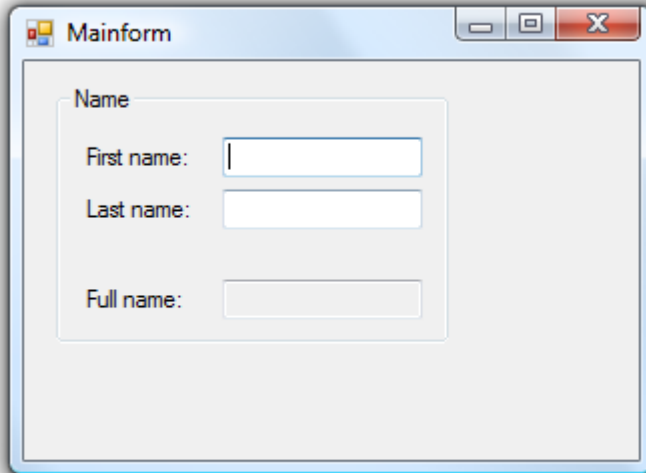




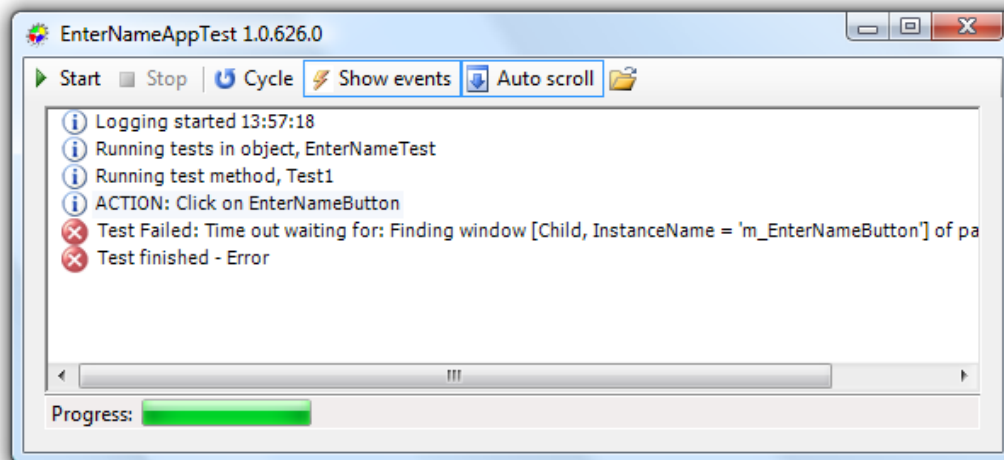
Test Automation FX

UI test automation for .NET developers

name should be entered directly in the Mainform.



If we try to run the current test we will get an error since it cannot find the **Enter name** button any longer.



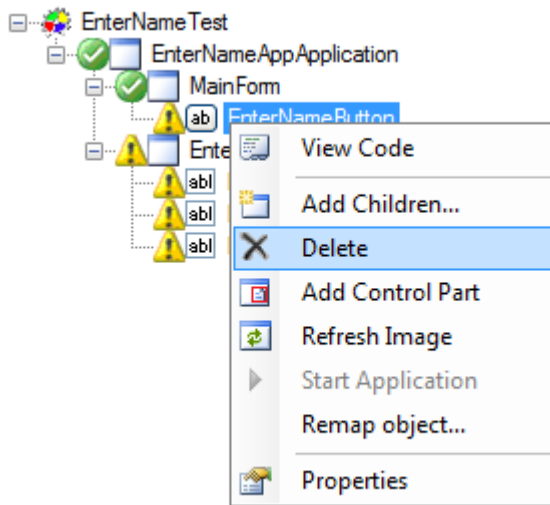
In this new version of the application the button is not needed so we will have to remove it from the test. Hence, our first adoption would be simply to delete it from the UIMap.





Test Automation FX

UI test automation for .NET developers



The next step is to also remove it from the test logic. If we removed it from the UIMap, Visual Studio will now show us all uses of this object and tell us that it cannot be found anymore.

```
[UITest ()]
public void Test1 ()
{
    EnterNameButton.Click ();
    Keyboard.SendKeys ("Tom");
    LastName.Click (31, 15);
    Keyboard.SendKeys ("Jones");
    FullName.VerifyProperty ("Text", "Tom Jones");
}
```

Since we do not want to use it anymore, we delete the call to `EnterNameButton.Click()`

```
[UITest ()]
public void Test1 ()
{
    FirstName.Click (44, 8);
    Keyboard.SendKeys ("Tom");
    LastName.Click (31, 15);
    Keyboard.SendKeys ("Jones");
    FullName.VerifyProperty ("Text", "Tom Jones");
}
```

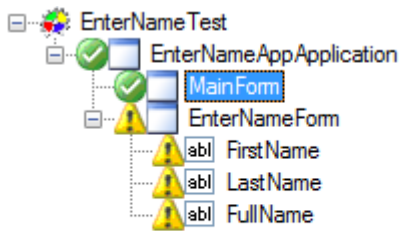
Running the test now would still give us an error since the name controls has been moved. The fact that these objects cannot be found is shown in the UIMap by the objects being marked with a warning icon.



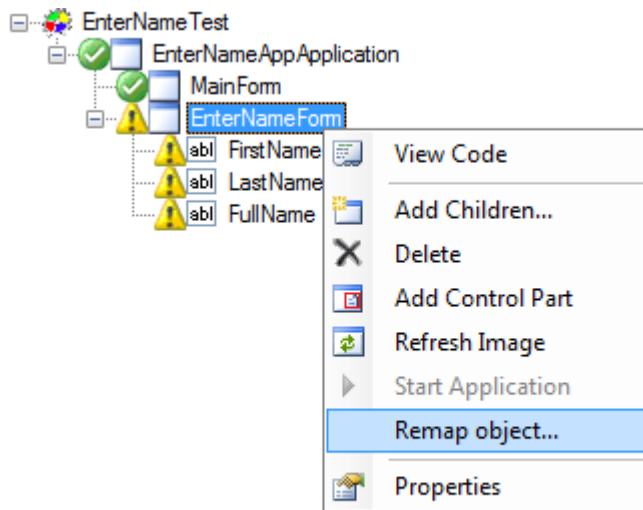


Test Automation FX

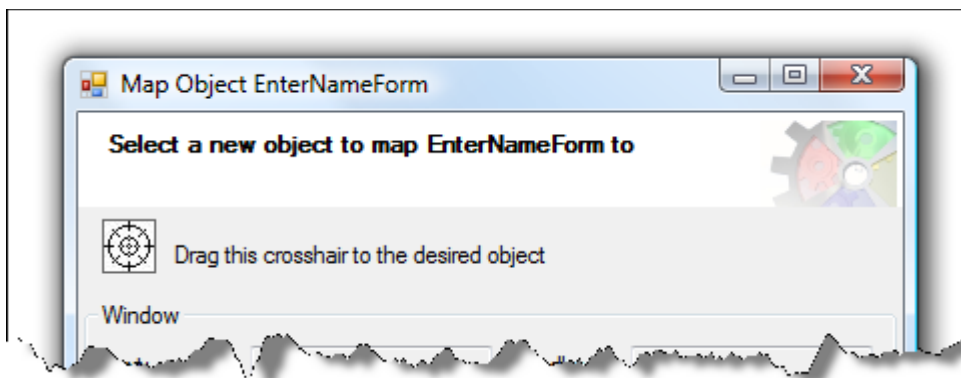
UI test automation for .NET developers



To get the test to start working again we now need to remap these controls. In this case we can do this by simply remapping the parent of the objects, i.e. we remap EnterNameForm to the groupbox containing the controls in the mainform. (We could of course also remap the objects one by one.)



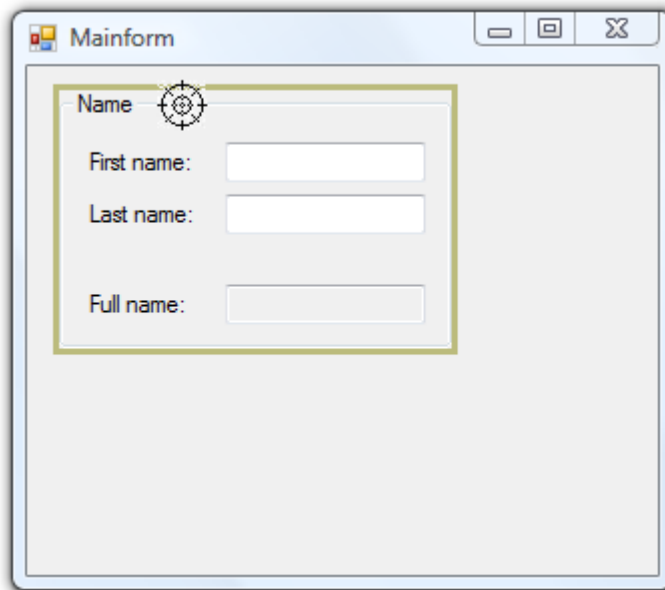
The remapping is a simple process of dragging a crosshair to the new user interface element constituting the new parent of the name controls.



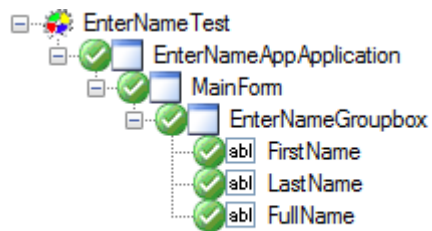


Test Automation FX

UI test automation for .NET developers



After the remapping the UIMap looks like this:



And the test is working again.

