

M-eux Test

Test Automation using Visual Studio User Guide

Abstract

This Getting Started Guide describes how to use M-eux Test for testing mobile applications using Visual Studio. This document is intended for Quality Assurance (QA) engineers and testers who wish to get acquainted with the functionalities of M-eux Test.

© Copyright 2013 Jamo Solutions N.V. No parts of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Jamo Solutions.

This document is provided for informational purposes only. Jamo Solutions makes no warranties as to the information in this document. The information contained herein is subject to change without notice.

All trademarks are the properties of their respective owners

Contents

Contents	3
Chapter 1: Introduction	6
1.1. M-eux Test overview.....	6
1.2. Where to go from here	7
Chapter 2: M-eux Test Overview	8
2.1. The mobile device agent	8
2.2. The device manager application	8
2.2.1. The connected devices.....	8
2.2.2. The connected testing tools.....	9
2.3. Visual Studio and the .Net Framework	9
Chapter 3: Getting Started.....	10
3.1. Starting M-eux Test.....	10
3.2. Creating a M-eux Test script	13
3.3. Creating a M-eux Test WM ROD script	15
3.4. Recording a Script	17
3.5. Learn GUI	19
3.6. Programming a script manually	20
3.6.1. IntelliSense	20
3.6.2. Change the name of a ScriptObject inside the UserScript/ObjectPool	20
3.7. ObjectPool.....	22
3.7.1. The root ScriptObjects	22
3.7.2. The ScriptObject hierarchy	24
3.7.3. ScriptObject Properties.....	25
3.7.4. Reuse an existing ObjectPool for another Userscript	27
3.8. Saving a script	32
3.9. Replaying a script	32
3.10. Distribute a script.....	34
3.11. Replaying a distributed script	35
3.12. MeuxExecuter Commandline Options	35
3.13. Batch execution	36

3.14. Results of a script, using the ResultManager.....	36
3.15. Logging statements.....	36
3.16. GlobalSucces Property	37
3.17. LogRunStatements Property.....	37
3.18. Exceptions	37
3.18.1. MeuxException	37
3.19. Run one script from another Script	38
Chapter 4: Advanced Options	45
4.1. The MoListViewCE object in Windows Mobile Standard Edition	45
4.2. Testing .Net Compact Framework Applications	46
4.2.1. Automatic activation of the .Net Compact Framework Extended support.	46
4.2.2. How to active the automatic .Net extended support?	46
4.2.3. Enabling extended support in a non-automatic way.....	48
4.2.4. Register Form-approach	49
4.2.5. Overview extended functionality.....	50
4.2.6. Getting and Setting Properties	51
4.2.7. Parameters and return values	52
4.2.8. Supported controls	53
Chapter 5: Troubleshooting.....	55
5.1. The Add-in Meux VS Addin failed to load in VS 2010.	55
Chapter 6: Using M-eux Test with Unit Test Projects.....	56
6.1. Create your Unit Test Project.....	56
6.2. Create your unit test class	60
6.3. Record your first Unit Test.....	61
6.4. Converting a Unit Test into a Performance Test.....	62
6.5. Adding your unit tests to a performance test project	64
6.6. Run your tests cases from Microsoft Test Manager	66
6.7. Run your tests cases from Microsoft Team Build	66
Chapter 7: Using M-eux Test in any .NET Project	67
7.1. Assemblies	67
7.2. Connect the .Net executable to the device manager	67
7.3. Programming automation steps	68

7.3.1. The APIScript class	68
7.3.2. Automation methods	69
7.3.3. Object Pool	69
7.4. Descriptive programming	71
Chapter 8: Summary	73

Chapter 1: Introduction

This guide is the user's guide for the Visual Studio part of 'M-eux Test'. The product 'M-eux Test' uses Microsoft's Visual Studio to create and maintain automation scripts for testing against mobile devices.

In order to get used to the Visual Studio User Interface, please consult Microsoft's provided help and documentation resources. This guide will focus on the additional functionality in order to create test cases against mobile devices.

We have made every attempt possible in making the instructions in this guide as clear as possible. However, we recognize that we are unable to cover everything in a single guide. Should you require further assistance, please do not hesitate to visit our www.jamosolutions.com website or to contact our support team at support@jamosolutions.com.

1.1. M-eux Test overview

To test mobile applications using M-eux Test, you need three main components. The **mobile device** hosts the application that you want to test. You use a **Visual Studio** in which you write and execute your scripts. Finally, the **M-eux Test Device Manager** is the core of our application and acts as the gateway between the mobile devices and scripting environment.

M-eux Test currently supports the following products:

- **Mobile Device:** Mobile devices can only connect to the device manager if the agent is installed on the device. The device needs to be connected to the PC, on which Visual studio and the device manager are installed, by a USB cable or a WIFI-connection.
- **Device Manager:** The device manager application manages the connection between the devices and Eclipse.
- **Visual Studio:** version 2005, 2008, 2010, 2012 and 2013 are supported

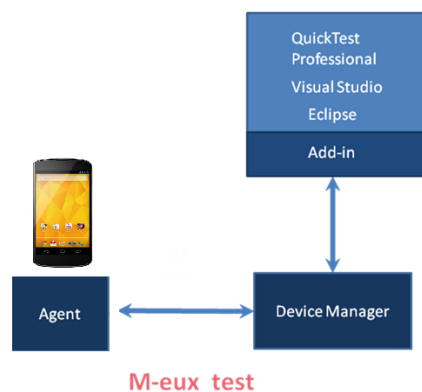


Figure 1: M-eux Test Architecture

1.2. Where to go from here

If you want to further explore the features of M-eux Test, please visit our website at www.jamosolutions.com for further information on the features of M-eux test.

Chapter 2: M-eux Test Overview

In order to test applications running on mobile devices, you need three components:

- **The mobile device agent** running on a mobile device or emulator connected with a USB cable or WIFI to the PC. In case of a USB cable, the connection is managed by ActiveSync from Microsoft.
- **The DeviceManager application.** This application is running on the PC and connects the mobile device with Visual Studio.
- **Visual Studio** for creating and maintaining the scripts.

2.1. The mobile device agent

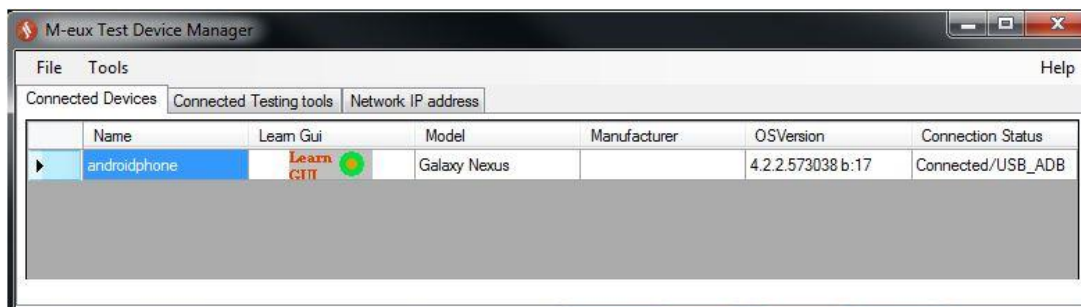
Mobile devices can only connect to the device manager if the agent is installed on the device. The device needs to be connected to the PC, on which Visual studio and the device manager are installed, by a USB cable or a WIFI-connection.

2.2. The device manager application

The device manager application manages the connection between the devices and Eclipse. The device manager application lists three tables:

- **The connected devices**
- **The connected testing tools**

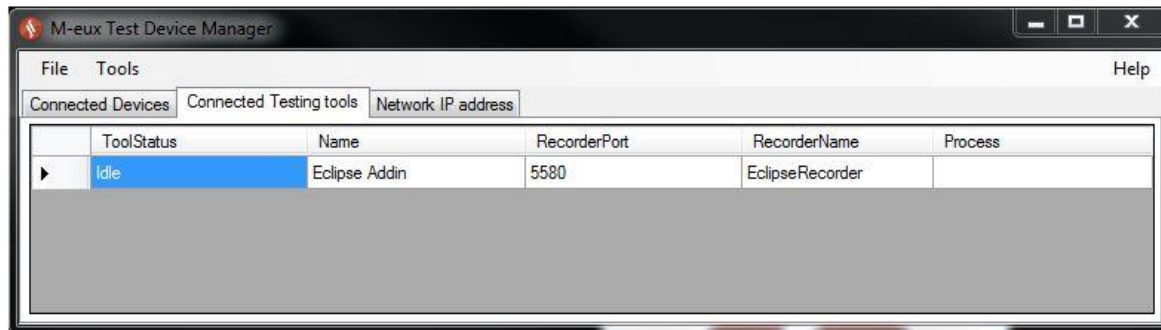
2.2.1. The connected devices



	Name	Learn Gui	Model	Manufacturer	OSVersion	Connection Status
▶	androidphone	Learn GUI	Galaxy Nexus		4.2.2.573038 b:17	Connected/USB_ADB

This table lists all connected devices. The first column displays the name of the device. This name is used to identify the device. You have to make sure that if you connect multiple devices, that each device has a unique name.

2.2.2. The connected testing tools



The screenshot shows the 'M-eux Test Device Manager' window. It has a menu bar with 'File', 'Tools', and 'Help'. Below the menu bar are three tabs: 'Connected Devices', 'Connected Testing tools', and 'Network IP address'. The 'Connected Testing tools' tab is active, displaying a table with the following data:

	ToolStatus	Name	RecorderPort	RecorderName	Process
▶	Idle	Eclipse Addin	5580	EclipseRecorder	

This table lists all connected testing tools to the device manager. The table will display the connected Visual Studio session and its status. The status can be 'recording', 'idle' or 'replaying'.

See [Appendix A](#) for more Information about the device manager.

2.3. Visual Studio and the .Net Framework

Visual studio is a development environment for making desktop and web-applications. Multiple programming languages are supported. The "M-eux Test"-extension focuses only on the C# .Net programming language. The following sections give a brief overview of prerequisite knowledge for making "M-eux Test"-scripts.

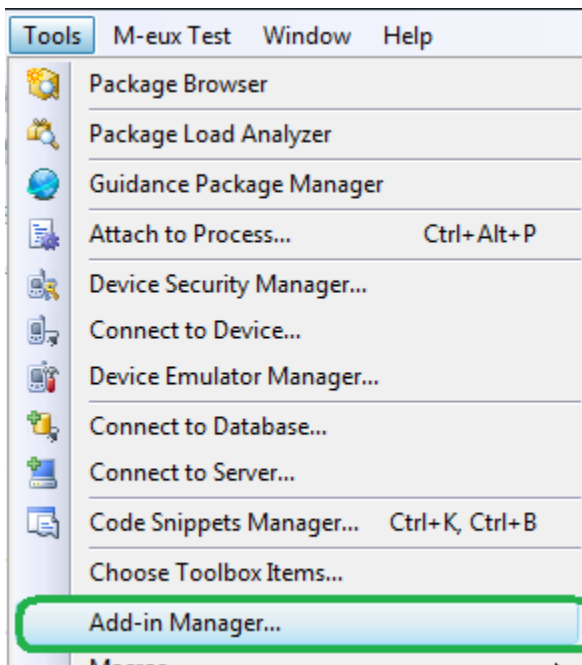
Please refer to Microsoft's documentation for a complete overview:
[http://msdn.microsoft.com/nl-be/library/52f3sw5c\(en-us\).aspx](http://msdn.microsoft.com/nl-be/library/52f3sw5c(en-us).aspx)

M-eux Test do support Microsoft Visual Studio professional or later

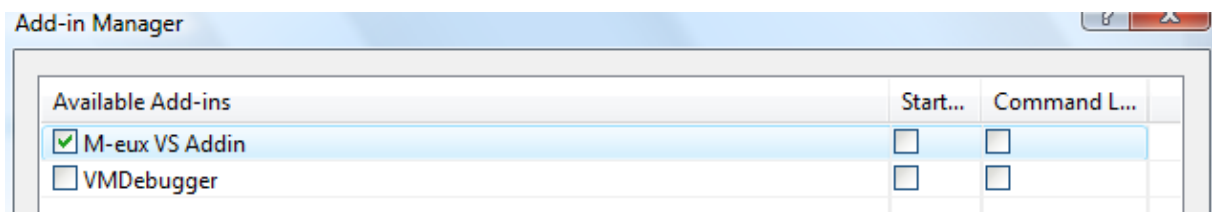
Chapter 3: Getting Started

3.1. Starting M-eux Test

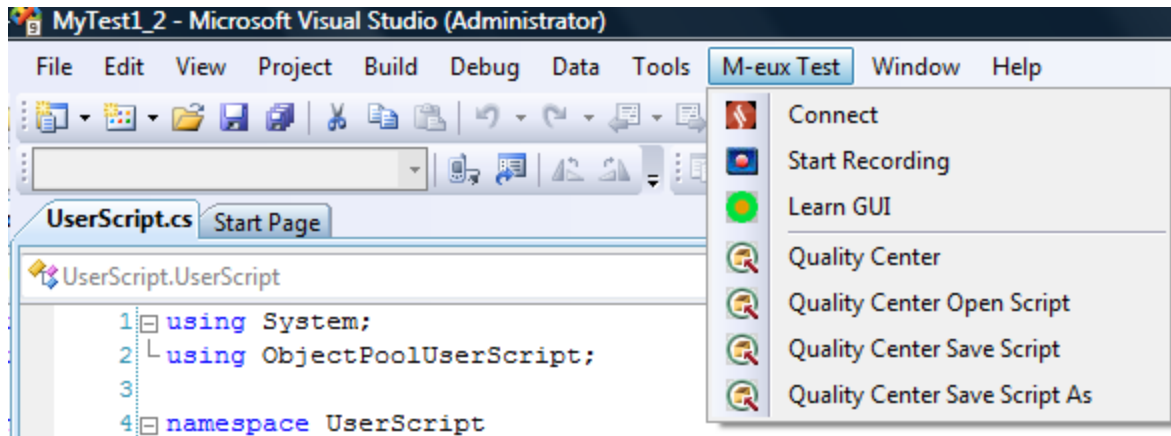
1. The device manager needs to be started before Visual Studio is started. If the device manager program is not running, the M-eux Test package in Visual Studio cannot be activated.
2. Launch the Agent on your mobile device.
3. Start Visual Studio. From the tools menu select “Add-in Manager”.



The Add-in Manager Window will appear:

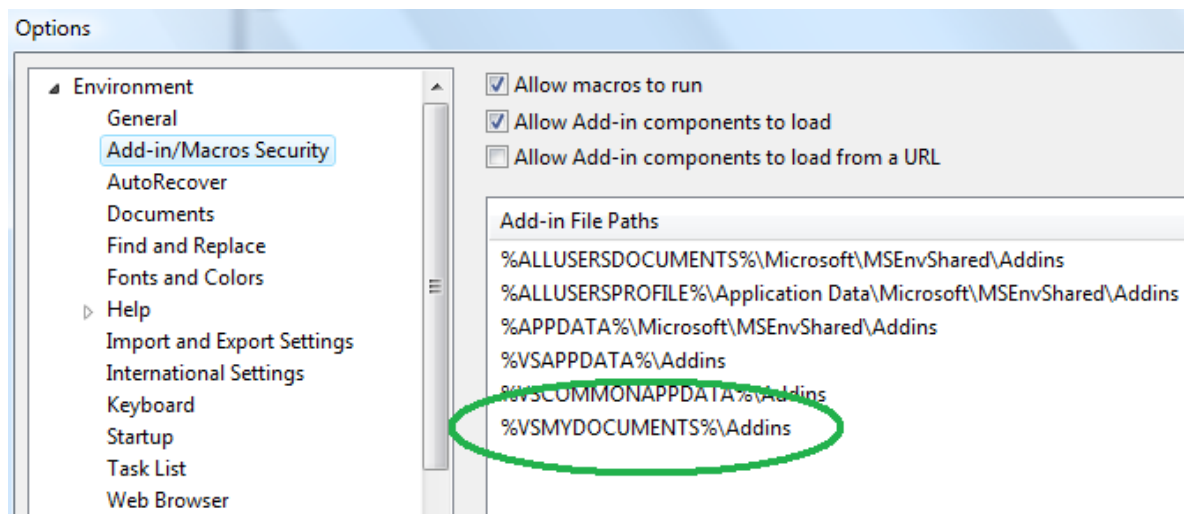


Check the M-eux VS Addin option. The M-eux Test Menu will be added to Visual Studio :

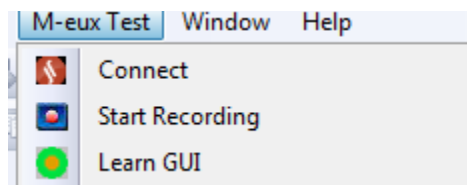


Troubleshooting:

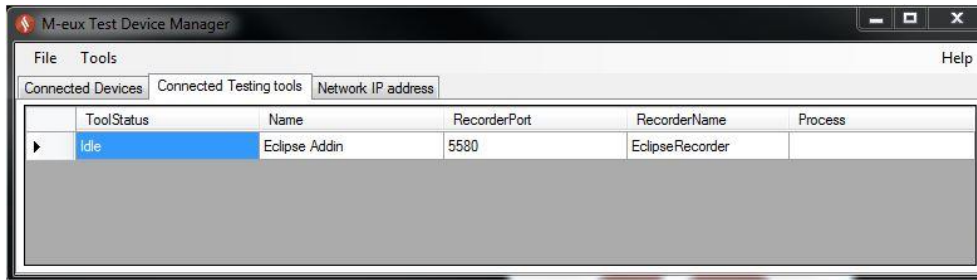
If the M-eux Test menu does not get added please verify in Tools->Options->Environment->Add-in /Macros Security if you have the "%VSMYDOCUMENTS%\Addins" path listed in Add-in File Paths.



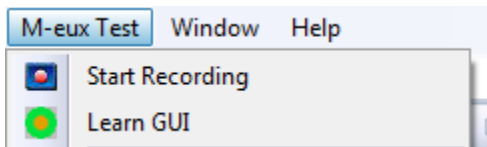
Select "Connect" from the " M-eux Test Menu" to connect Visual Studio to the device manager.



If the device manager was not yet running it will be started. You will see the Visual Studio Add-in appear in the "Connected Testing Tools" tab of the device manager:



Once the tool is connected the “Connect” option will no longer be available in the “M-eux Test Menu” until the device manager is closed.



While you are using the tool, take care of the following limitations:

- Once the automatic start check box has been checked, the agent will start automatically the next time the device is reset. If the agent is running automatically and there is a need to change the connection details, for example the connected PC name, then the agent needs to be stopped and started again. The agent window will open and the connection information can be changed.
- To stop a running agent, one can use the ‘m-eux control panel’ application. Open the File explorer on the device, navigate to the agent installation directory and select the ‘agenttcp’ application. Following screen is displayed:

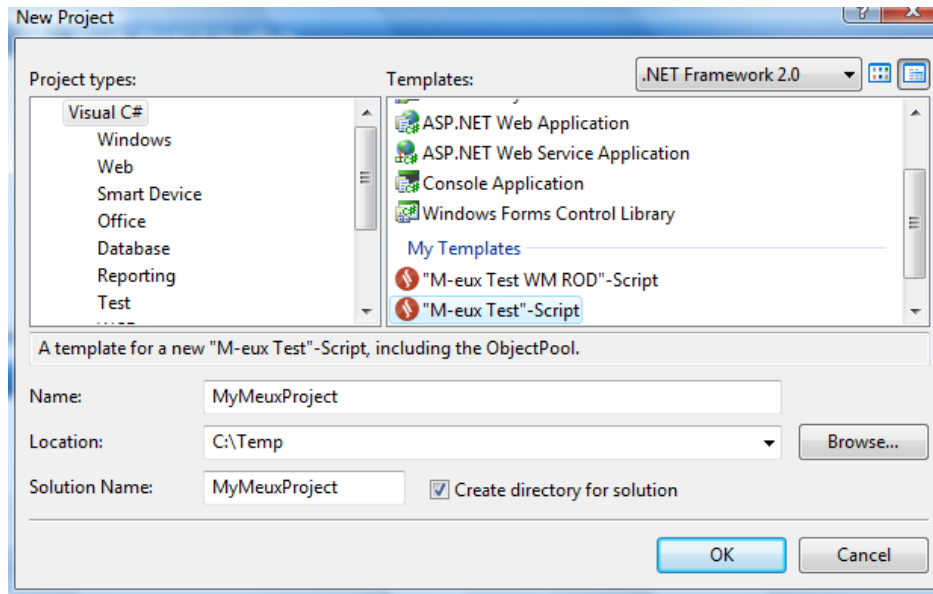


- The window shows information about the agent: the running state, the version of the agent, the name of the PC to which the agent connects, the connection number and if automatic startup is enabled. This information is updated by selecting the left menu 'Refresh'. Using the right menu 'Menu', the agent can be stopped or started.

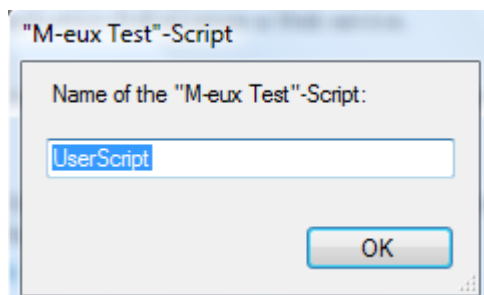
3.2. Creating a M-eux Test script

Start the environment as described in the previous section.

1. Create a new project inside Visual Studio:
File → New → Project...
2. In the dialog select the template: Visual C# → "M-eux Test"-Script



3. Give the solution a Name and a Location and press OK
4. Specify a unique name for the script.

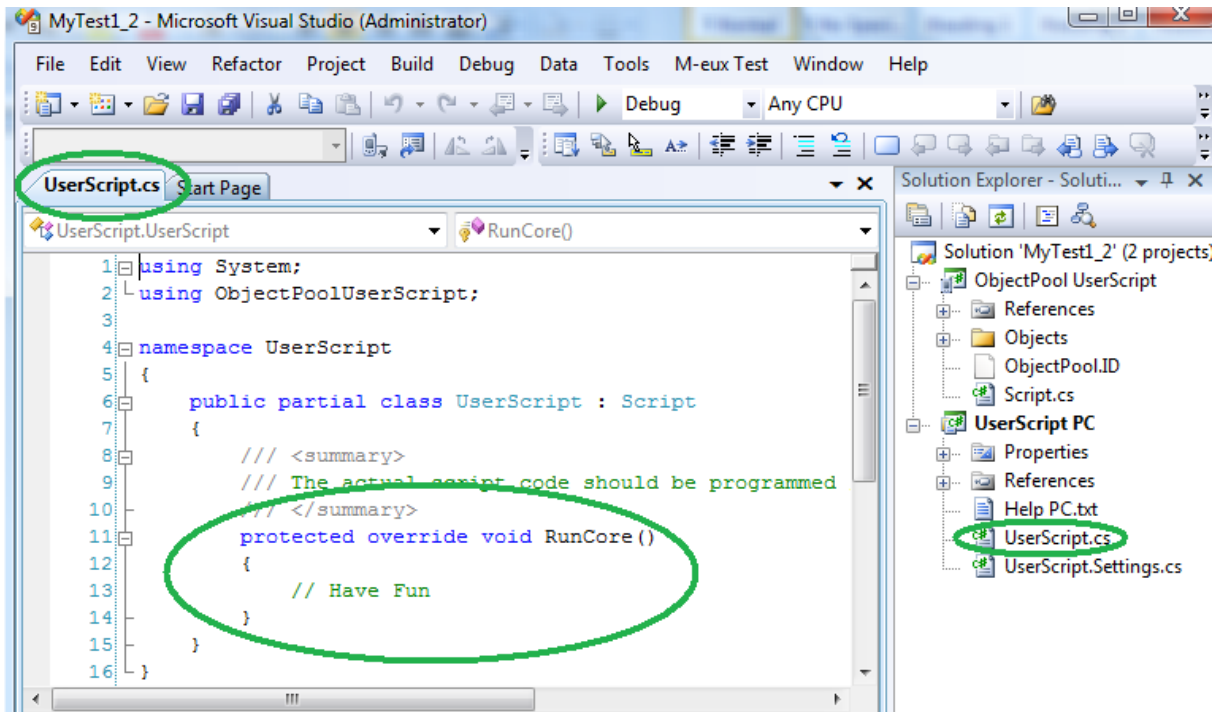


Note: it is very important to specify a unique name for your script, this will ease both the sharing of the ObjectPool between other scripts and calling one script from another.

5. A new Solution with 2 projects will appear:
 - a. ObjectPool: this project will contain all the objects that can be approached for automation testing, together with their descriptions.
 - b. UserScript PC: represents the actual script which will contain the automation logic. The project contains only two “.cs”-files:
 - i. “UserScript.cs”: opened by default after the template has been loaded.
 - ii. “UserScript.Settings.cs”
6. You can start making a script. There are 2 approaches for generating a script:
 - a. Record a script by recording manual operations on the device.

- b. Learn the user interface of the device and program the script yourself.

Both possibilities will be explained in the next chapters.

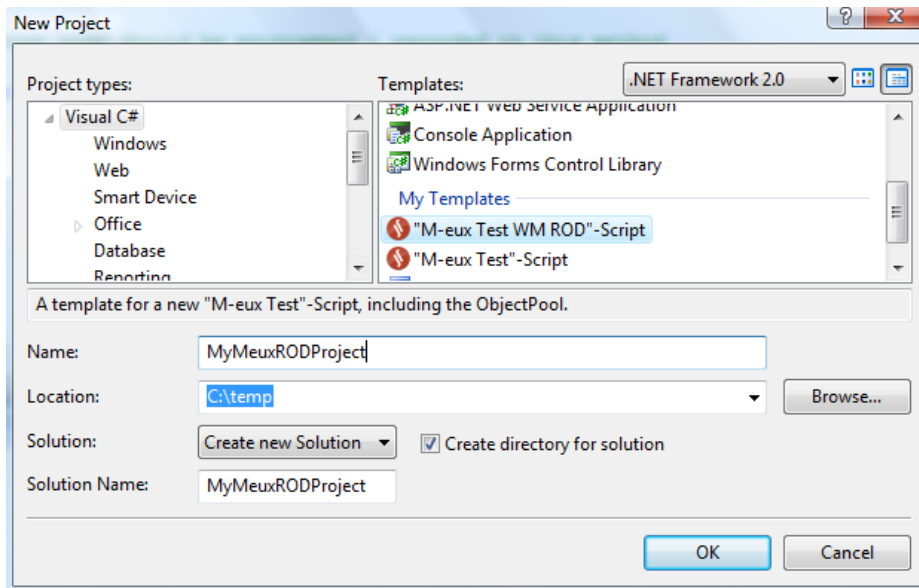


3.3. Creating a M-eux Test WM ROD script

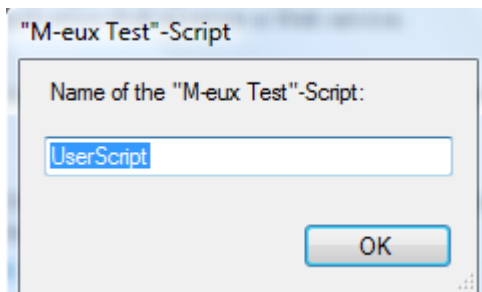
Start the environment as described in the previous section.

Please note that RoD is only available for Visual Studio 2005 and 2008.

1. Create a new project inside Visual Studio:
File → New → Project...
2. In the dialog select the template: Visual C# → "M-eux Test WM ROD"-Script

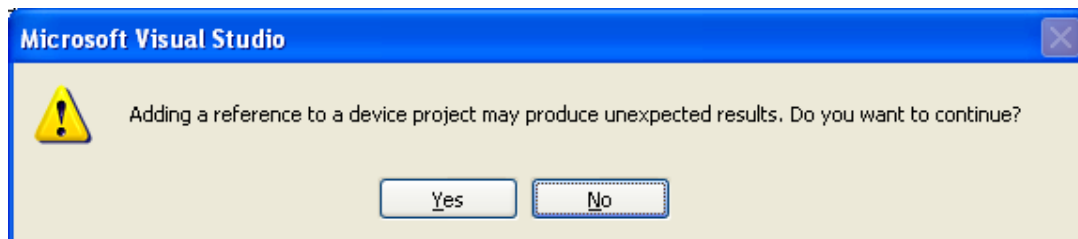


3. Give the solution a Name and a Location and press OK
4. Specify a unique name for the script.



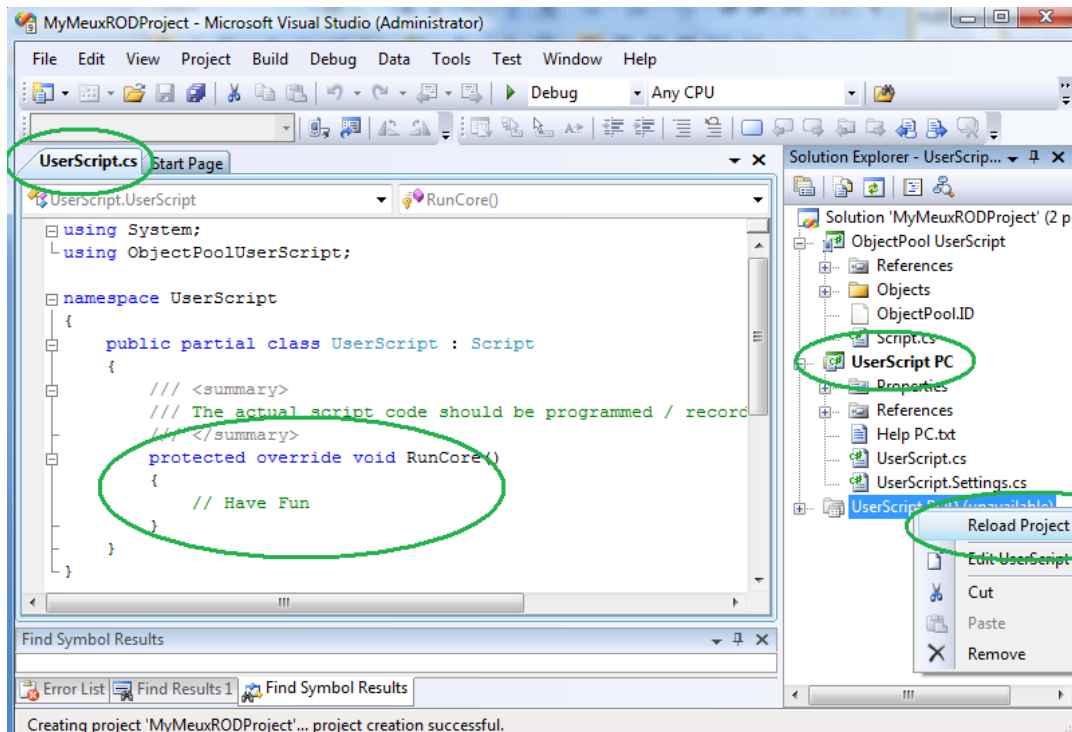
Note: it is very important to specify a unique name for your script, this will ease both the sharing of the ObjectPool between other scripts and calling one script from another.

5. Visual Studio 2005 users may get following warning, select "Yes"



6. A new Solution with 3 projects will appear:
 - a. ObjectPool: this project will contain all the objects that can be approached for automation testing, together with their descriptions.

- b. UserScript PC: represents the actual script which will contain the automation logic.
The project contains only two “.cs”-files:
 - iii. “UserScript.cs”: opened by default after the template has been loaded.
 - iv. “UserScript.Settings.cs”
- c. UserScript ROD: Right click on this project and choose “Reload Project”.



7. You can start making a script. There are 2 approaches for generating a script:
- a. Record a script by recording manual operations on the device.
 - b. Learn the user interface of the device and program the script yourself.

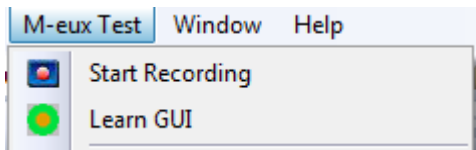
Both possibilities will be explained in the next chapters.

3.4. Recording a Script

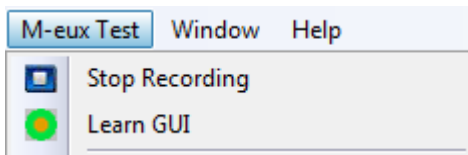
1. Create a new script as described earlier.
2. Make sure the device is connected to the Device Manager.



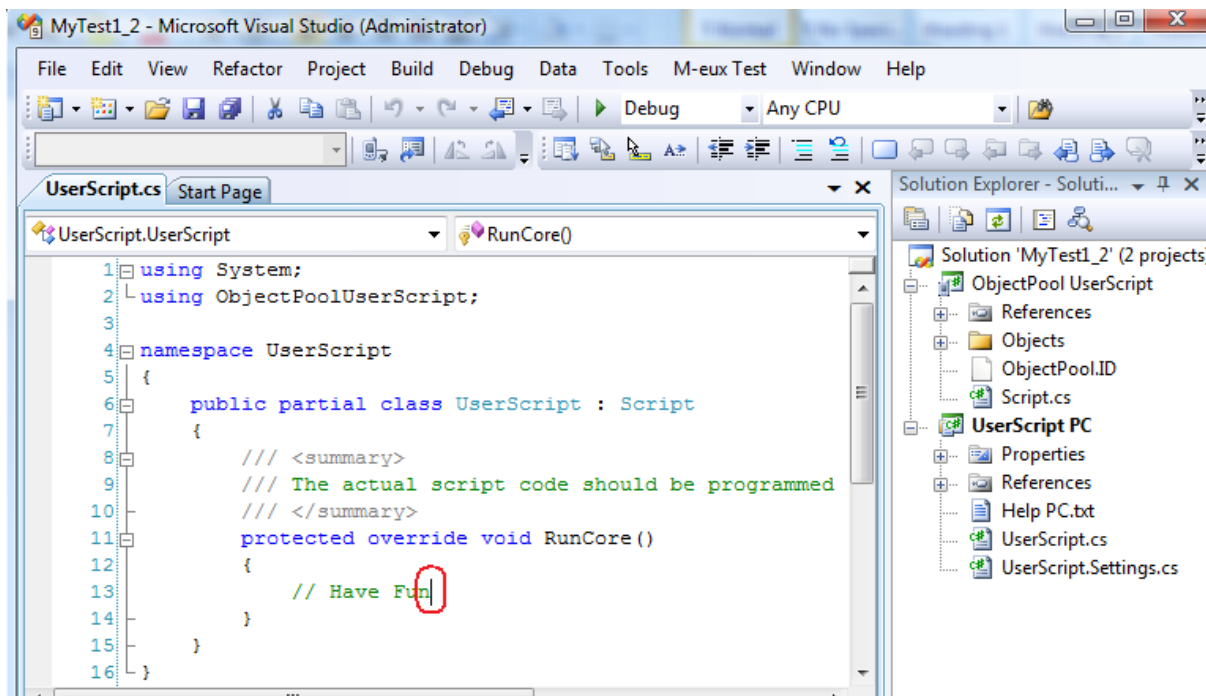
- From the “M-eux Test Menu” select : “Start Recording”



- While recording the “Stop Recording” option will be made available.

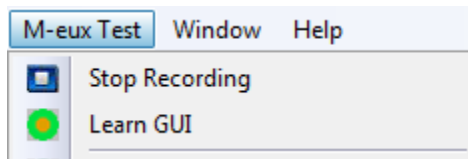


- Place the cursor where recorded statements should be generated, this should be somewhere inside the RunCore method inside the UserScript class:



- Tap on some Gui objects on the connected mobile device. (For example in the file explorer window)

7. Select “Stop Recording” from the “M-eux Test Menu” when finished.



The resulting script can look like this:

```

/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    // Have Fun
    hTC_P3470.file_Explorer.aTL_01F66680.sysListView32.Select("Temp");
    hTC_P3470.file_Explorer.omhoog_Menu.Select("Omhoog");
    hTC_P3470.file_Explorer.omhoog_Menu.Select("Menu");
    hTC_P3470.file_Explorer.omhoog_Menu.mNU.Select("My Documents");
}

```

3.5. Learn GUI

The Device Manager object will be available by default in the ObjectPool. If it is removed somehow, the device manager object can be learned at all times by the objectPool in Visual Studio. Select “Learn GUI” from the M-eux Test Menu.



The cursor will change into a hand and Visual Studio will be minimized.

In order to add the objects of the device, move the hand cursor to the device manager window and click on the cell ‘Learn GUI’ of the ‘connected devices’ table. The screen on the corresponding device will be learned and appear inside the ObjectPool of the Visual Studio Solution.

Right clicking will cancel the Learn GUI.



Note that the upper toolbar, called MoTaskBar will not be learned in this mode. Only the active foreground window is learned. Popped-up menus are also not learned in this mode.

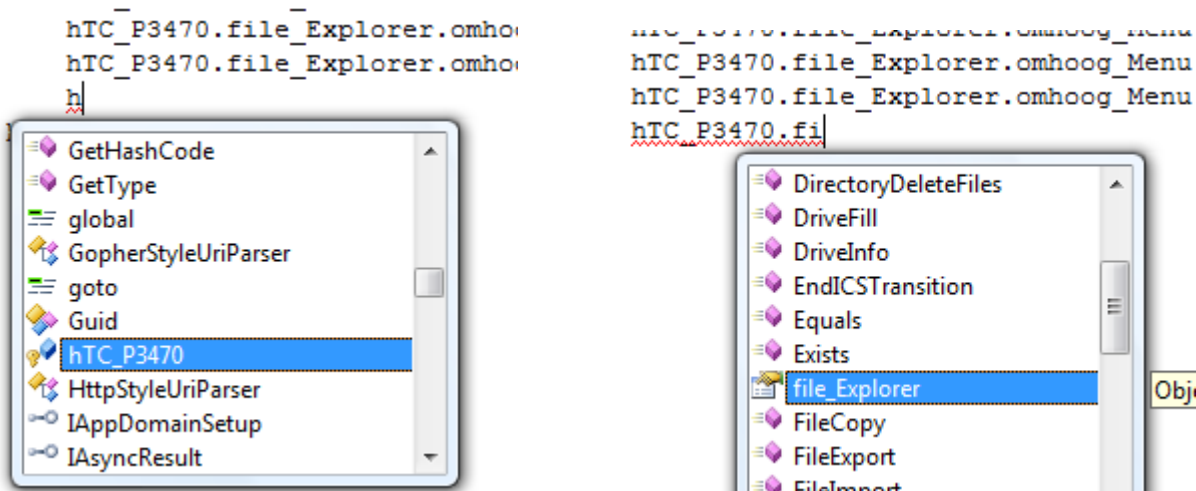
After clicking, wait until the following message disappears:

Do not interact with your application while its objects are being added to the objectPool.

3.6. Programming a script manually

3.6.1. IntelliSense

For programming, all visual studio tools are at your disposal, a very useful feature will be “intellisense”. The objects in the objectPool can be easily accessed, and their methods are also available for Autocompletion.



3.6.2. Change the name of a ScriptObject inside the UserScript/ObjectPool

The initial script can look like this:

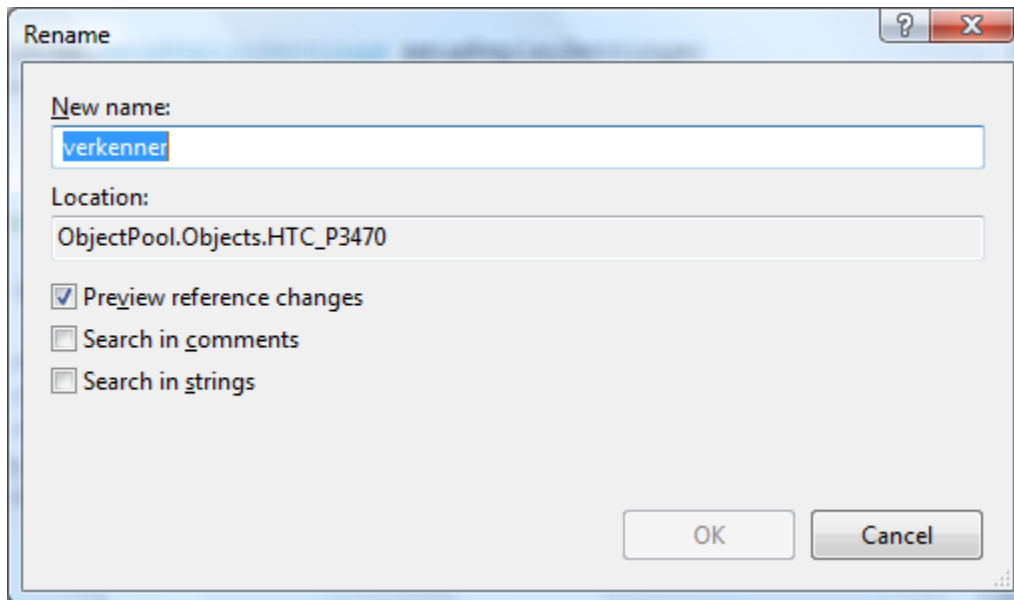
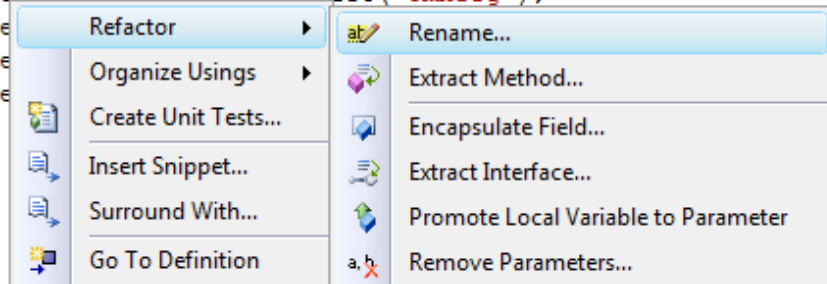
```
/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore ()
{
    hTC_P3470.verkenner.atL_01F66680.sysListView32.Select("Temp");
    hTC_P3470.verkenner.omhoog_Menu.Select("Omhoog");
    hTC_P3470.verkenner.omhoog_Menu.Select("Menu");
    hTC_P3470.verkenner.omhoog_Menu.mNU.Select("My Documents");
    hTC_P3470.verkenner.omhoog_Menu.Select("Omhoog");
}
```

Right click on the name of an object you want to change inside the script and select the Refactor -> rename option:

```

/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    hTC_P3470.verkenner.aTL_01F66680.sysListView32.Select("Temp");
    hTC_P3470.verkenner.omhoog_Menu.Select("Omhoog");
    hTC_P3470.verkenner.omhoog_Menu.mNU.Select("My Documents");
    hTC_P3470.verkenner.omhoog_Menu.Select("Omhoog");
}

```



Give it the name you want. Press OK. Press Apply. The renaming should be

```

/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    hTC_P3470.file_Explorer.aTL_01F66680.sysListView32.Select("Temp");
    hTC_P3470.file_Explorer.omhoog_Menu.Select("Omhoog");
    hTC_P3470.file_Explorer.omhoog_Menu.Select("Menu");
    hTC_P3470.file_Explorer.omhoog_Menu.mNU.Select("My Documents");
    hTC_P3470.file_Explorer.omhoog_Menu.Select("Omhoog");
}

```

3.7. ObjectPool

The ObjectPool-project manages all automation objects that can be used inside the userscript, together with their descriptions. All the test objects' classes that can occur in the objectPool have a common parent-class: "ScriptObject", and will be referred to as ScriptObjects.

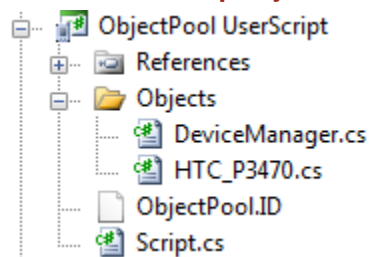
ScriptObjects can get in the object pool:

- By recording
- By learning the Gui

The objects are organized in a tree-structure. Several root-objects can contain child-objects. The hierarchies of objects represent the hierarchy of the corresponding objects on the device.

Note: the ObjectPool project should always contain the file ObjectPool.ID, never touch this file. The absence of the file inside the ObjectPool-project will disable recording and learn GUI functionality.

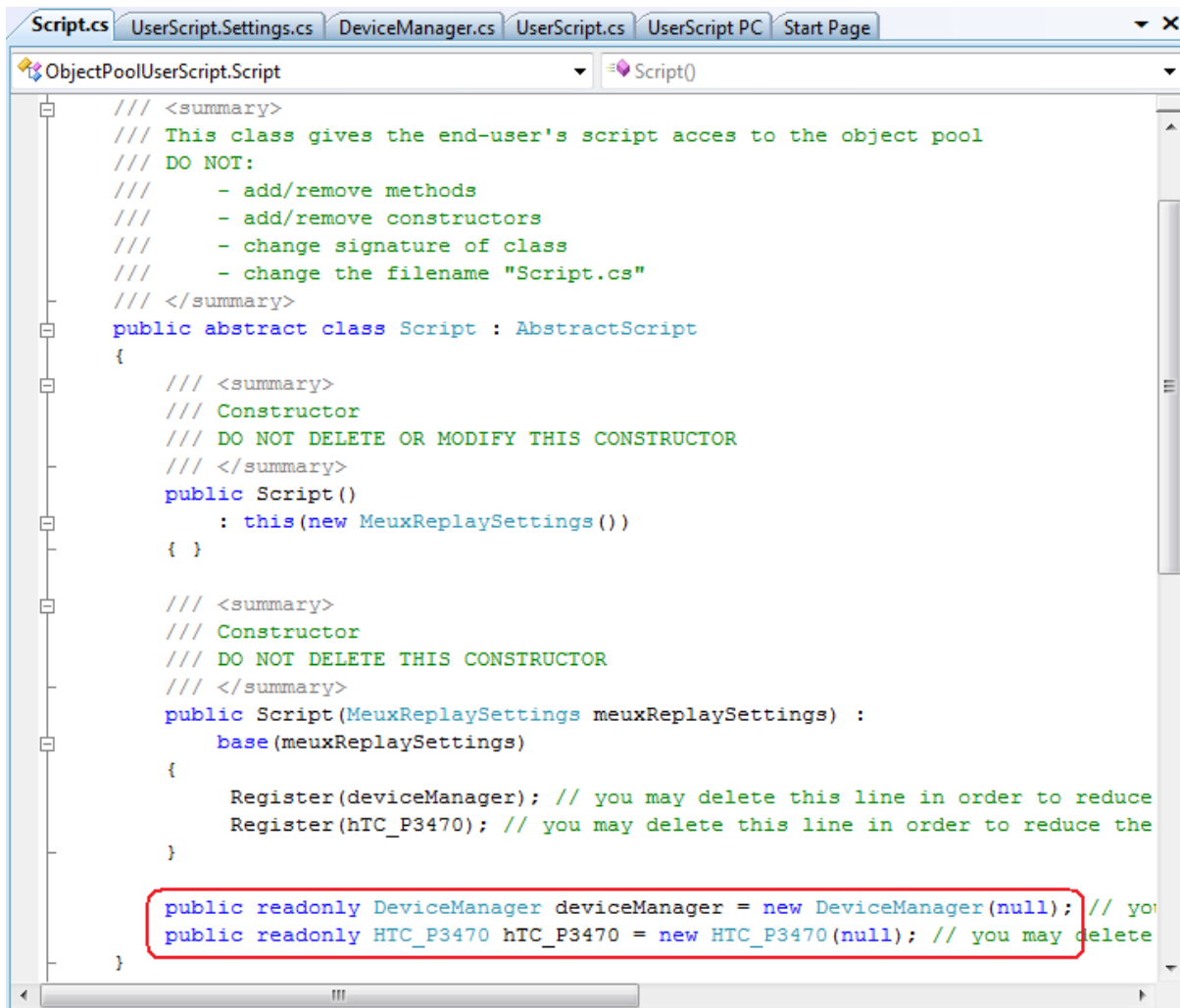
3.7.1. The root ScriptObjects



For each root ScriptObject, there is a corresponding ".cs"-file in the Objects folder. These contain the class definition of those objects. In the example there are 2 root-objects:

- the DeviceManager
- an object representing a mobile device (HTC_P3470)

To be able to access those root-ScriptObjects in the userscript, a reference to the instances of these Root-ScriptObjects is provided inside the file "Script.cs".



A detailed definition of a root-ScriptObject can be found in the corresponding ".cs"-file.

The root-ScriptObjects need to be registered inside the Script class:

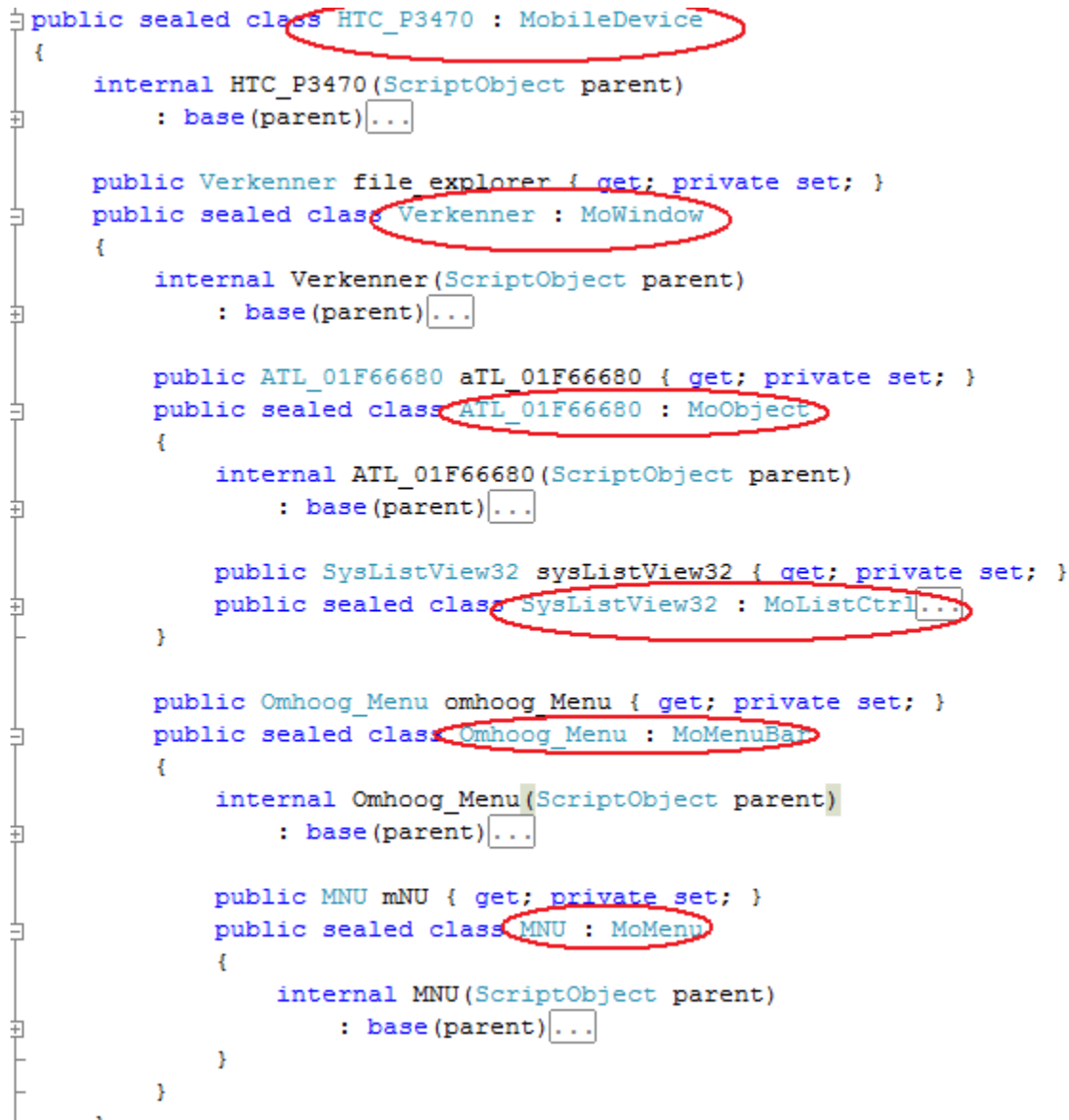
```

/// <summary>
/// Constructor
/// DO NOT DELETE THIS CONSTRUCTOR
/// </summary>
public Script(MeuxReplaySettings meuxReplaySettings) :
    base(meuxReplaySettings)
{
    Register(deviceManager); // you may delete this line
    Register(hTC_P3470); // you may delete this line
}

```

3.7.2. The ScriptObject hierarchy

When opening the “.cs”-file of a root-ScriptObject a hierarchy can be seen:



```

public sealed class HTC_P3470 : MobileDevice
{
    internal HTC_P3470(ScriptObject parent)
        : base(parent) {...}

    public Verkenner file_explorer { get; private set; }
    public sealed class Verkenner : MoWindow
    {
        internal Verkenner(ScriptObject parent)
            : base(parent) {...}

        public ATL_01F66680 aTL_01F66680 { get; private set; }
        public sealed class ATL_01F66680 : MoObject
        {
            internal ATL_01F66680(ScriptObject parent)
                : base(parent) {...}

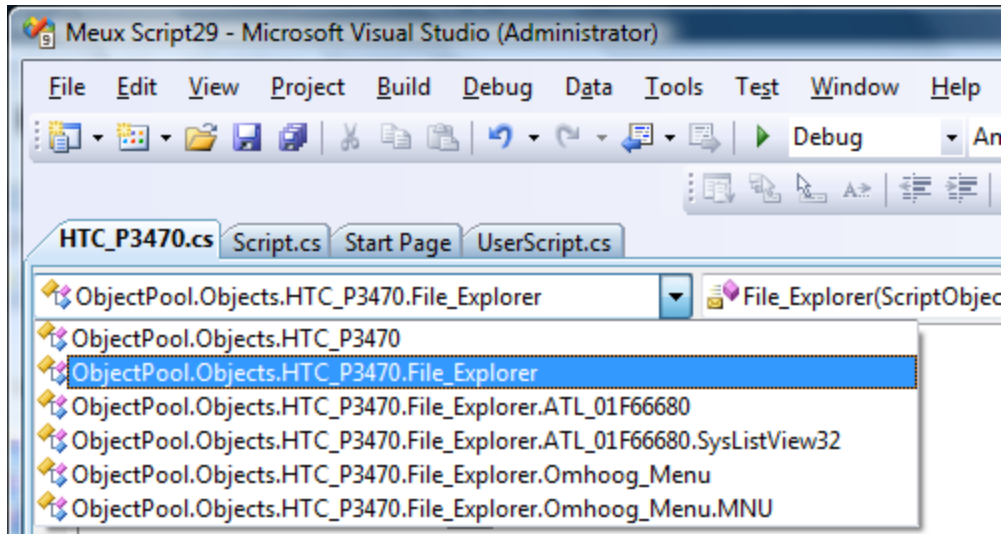
            public SysListView32 sysListView32 { get; private set; }
            public sealed class SysListView32 : MoListCtrl {...}
        }

        public Omhoog_Menu omhoog_Menu { get; private set; }
        public sealed class Omhoog_Menu : MoMenuBar
        {
            internal Omhoog_Menu(ScriptObject parent)
                : base(parent) {...}

            public MNU mNU { get; private set; }
            public sealed class MNU : MoMenu
            {
                internal MNU(ScriptObject parent)
                    : base(parent) {...}
            }
        }
    }
}

```

To get a better overview, the dropdown box in the upper left corner can be pressed:



3.7.3. ScriptObject Properties

Each ScriptObject in the ObjectPool will have a predefined set of properties available. Some of these properties can be used to identify the object. Only the properties that are used for the identification of an object can be seen in the ObjectPool.

If there are too many identifying properties for an object, the executing-userscript may not be able to find the object matching all those descriptions.

If there are too few identifying properties, the executing-userscript may find multiple GUI-objects matching the description properties.

The set of identifying properties are set by default, depending the object class.

To see the properties that are used to identify an object, make sure the whole file is expanded (press the + signs on the left of the code file). Use the dropdown box above to jump to the appropriate section in the file.

In the example, the details of the file_Explorer MoWindow are shown:

```
internal File_Explorer(ScriptObject parent)
: base(parent)
{
    PPClassname.Value = @"FEXPLORE";
    PPIId.Value = @"0";
    PPTitle.Value = @"Verkenner";

    AddDescriptionProperty(PPClassname);
    AddDescriptionProperty(PPIId);
    AddDescriptionProperty(PPTitle);

    aTL_01F66680 = new ATL_01F66680(this);
    omhoog_Menu = new Omhoog_Menu(this);
}
```

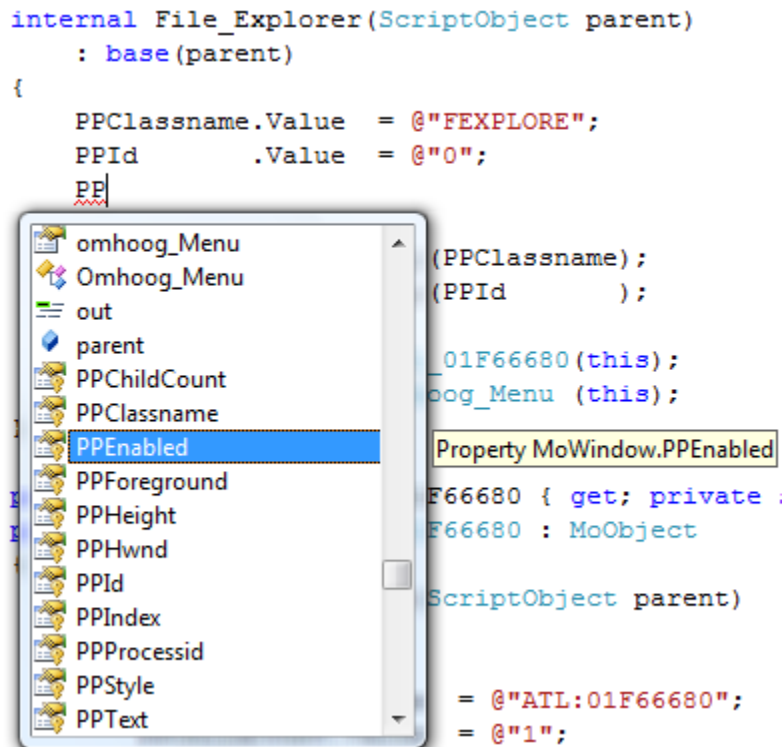
There are three identifying properties, the Classname, the Id and the Title of the window. To make the script able to run against multiple languages of devices, the Title property can be removed:

```
internal File_Explorer(ScriptObject parent)
: base(parent)
{
    PPClassname.Value = @"FEXPLORE";
    PPIId .Value = @"0";

    AddDescriptionProperty(PPClassname);
    AddDescriptionProperty(PPIId );

    atl_01F66680 = new ATL_01F66680(this);
    omhoog_Menu = new Omhoog_Menu (this);
}
```

To add other properties for identification, use the intellisense feature of visual studio. Start Typing “PP” and Visual Studio will show a list of all available properties for that kind of object:



Specifying regular expressions for the property values can be done like this:

```
PPClassname.Value = @"FEXPLORE";
PPIId .Value = @"0";
PPTitle .Value = @"File Explo.*";
PPTitle.RegExp = true;

AddDescriptionProperty(PPClassname);
AddDescriptionProperty(PPIId );
AddDescriptionProperty(PPTitle );
```

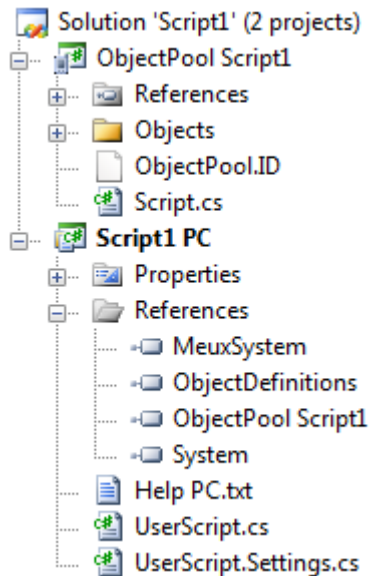
3.7.4. Reuse an existing ObjectPool for another Userscript

It can be useful for multiple projects to share their ObjectPool. This can be achieved by remapping the ObjectPool reference of the visual studio solution to another existing ObjectPool project.

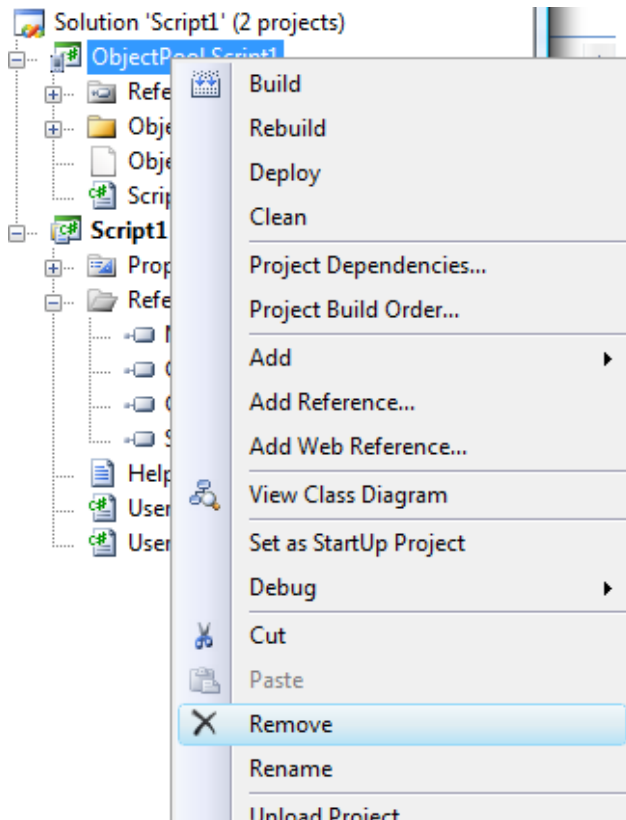
When projects are added to a solution, they are added by reference, no copy is taken.

Example

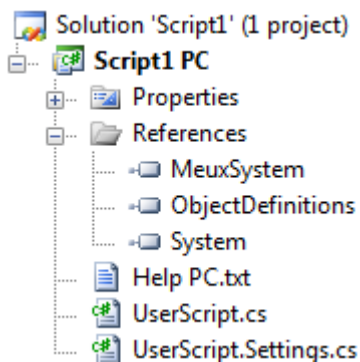
1. Create a new M-eux Script, or open an existing solution.



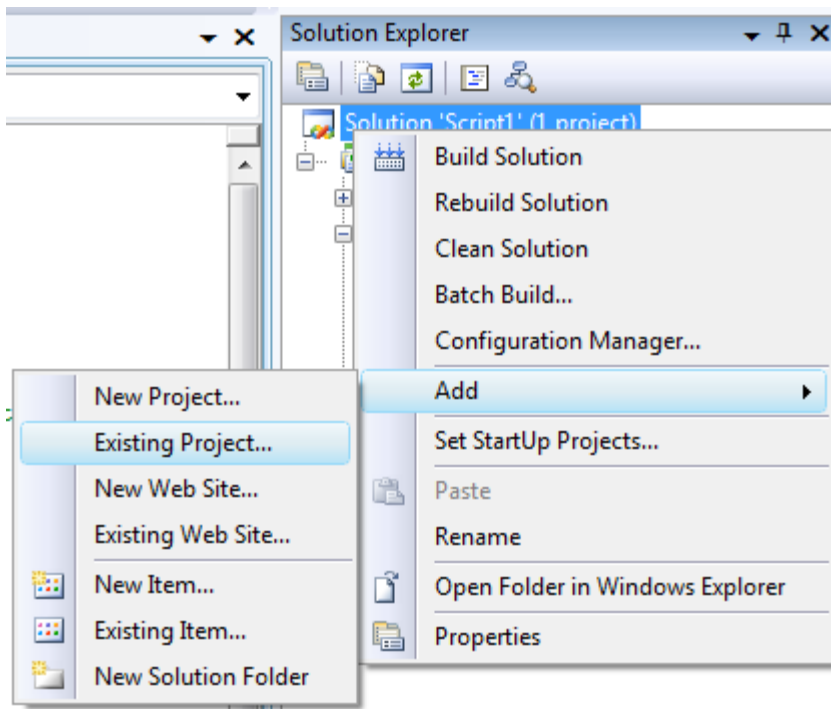
2. Remove the available ObjectPool-project.



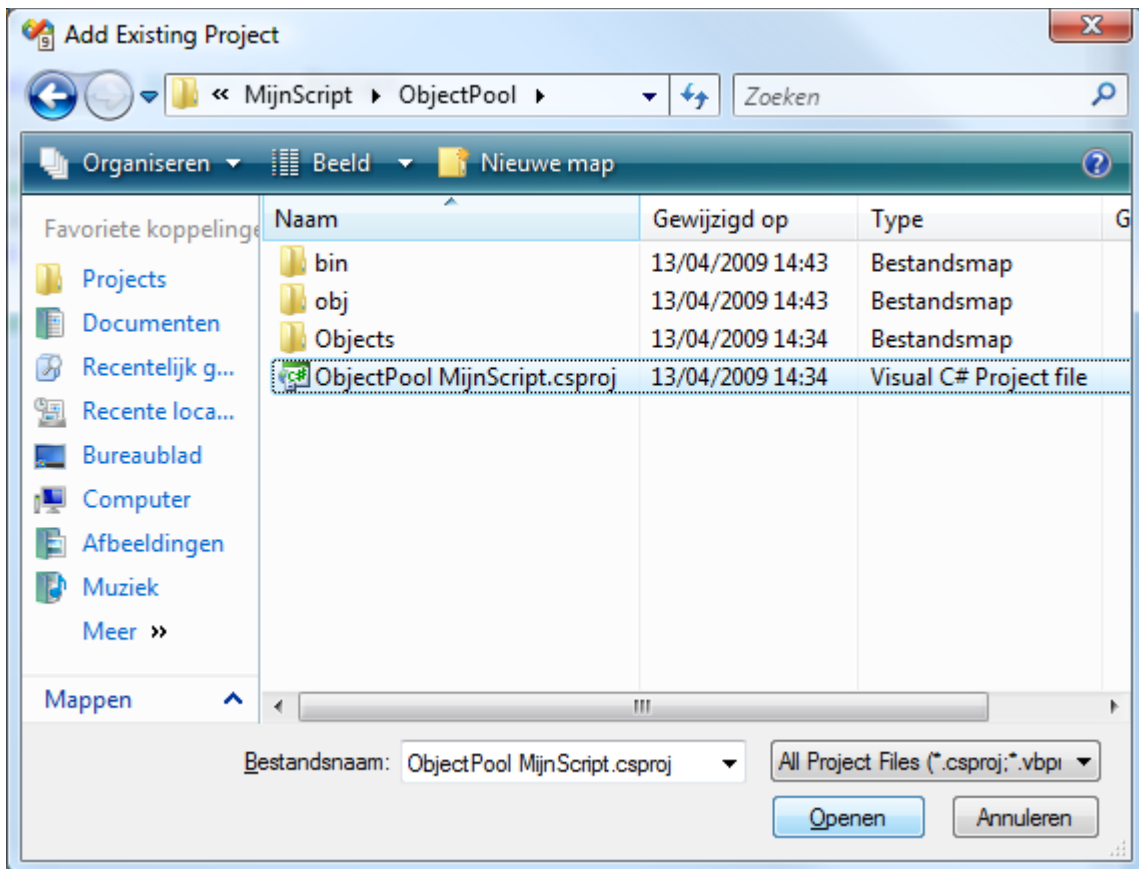
3. Note the ObjectPool reference is also removed from the UserScript-project.



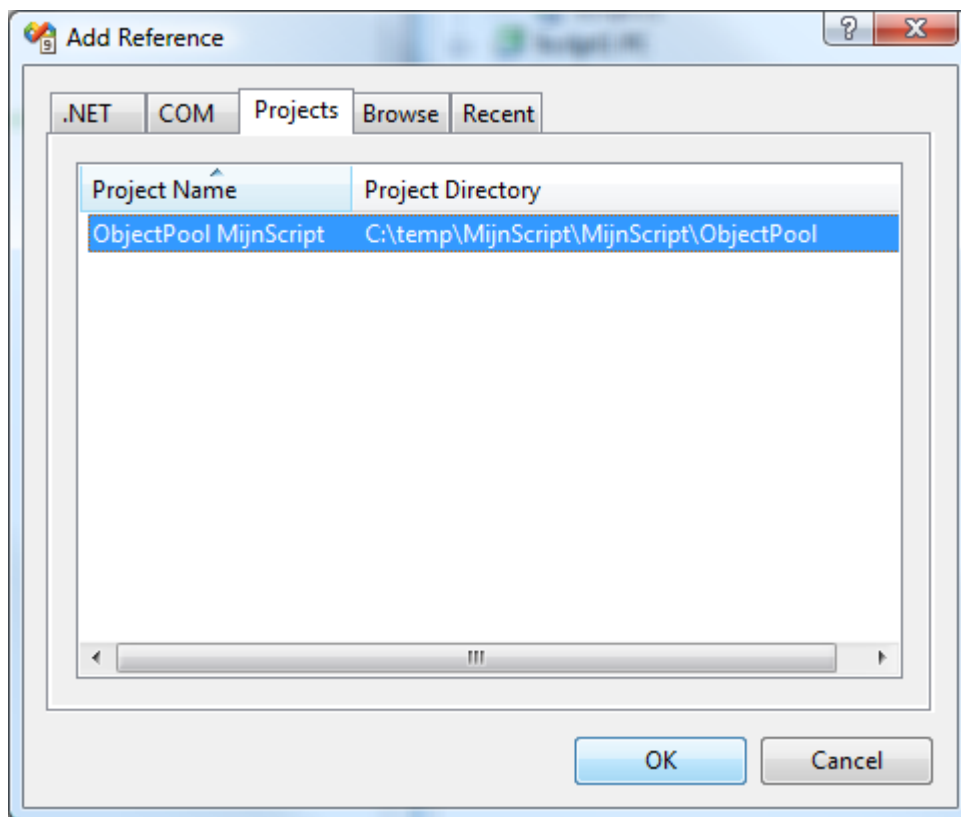
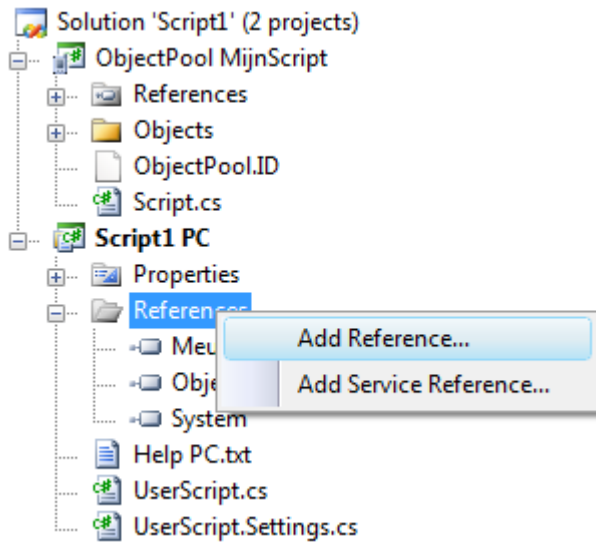
4. The removed ObjectPool still exists in the windows file explorer, if the removed ObjectPool-project is not needed anymore, the folder containing the removed ObjectPool-project can be deleted from the file system.
5. Add an existing ObjectPool to the solution

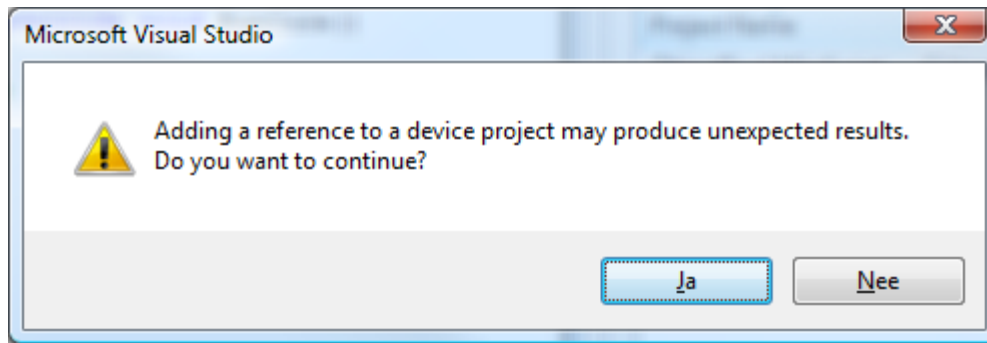


Navigate to and select the ObjectPool-project of another script.



6. Add a reference to the ObjectPool project in the references of the UserScript-project.





Select “Yes”

7. The userscript-project has now a reference to the other ObjectPool. Since all the ObjectPool-project’s classes are in another namespace then the original ObjectPool-project, the namespace reference inside userscript.cs has to be changed (Userscript.cs):

```
using System;
using ObjectPoolScript1;

namespace Script1
{
    public partial class Script1 : Script
    {
        /// <summary>
        /// The actual script code should be programmed / recorded in this method
        /// </summary>
        protected override void RunCore()
        {
            // Have Fun
        }
    }
}
```

Should become:

```
using System;
using ObjectPoolMijnScript;

namespace Script1
{
    public partial class Script1 : Script
    {
        /// <summary>
        /// The actual script code should be programmed / recorded in this method
        /// </summary>
        protected override void RunCore()
        {
            // Have Fun
        }
    }
}
```

The name of the new namespace can be found in “Script.cs” in the ObjectPool.

Note: visual studio is excellent for complex solution-, project- and references-management:

- Multiple projects can be added to a solution.
- References can be project references or dll references.

The “M-eux Test”-Tool only supports the default setting with only one ObjectPool project inside the solution, referenced by only one UserScript-project. When maintaining a more complex environment, only the replay of the script will be possible.

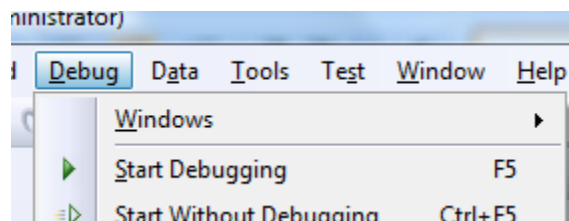
3.8. Saving a script

When the save menu options are used inside the “File”-menu in the menu bar of visual studio, only individual files can be saved, such as “.cs”-files, “.csproj”-files and “.sln files”.

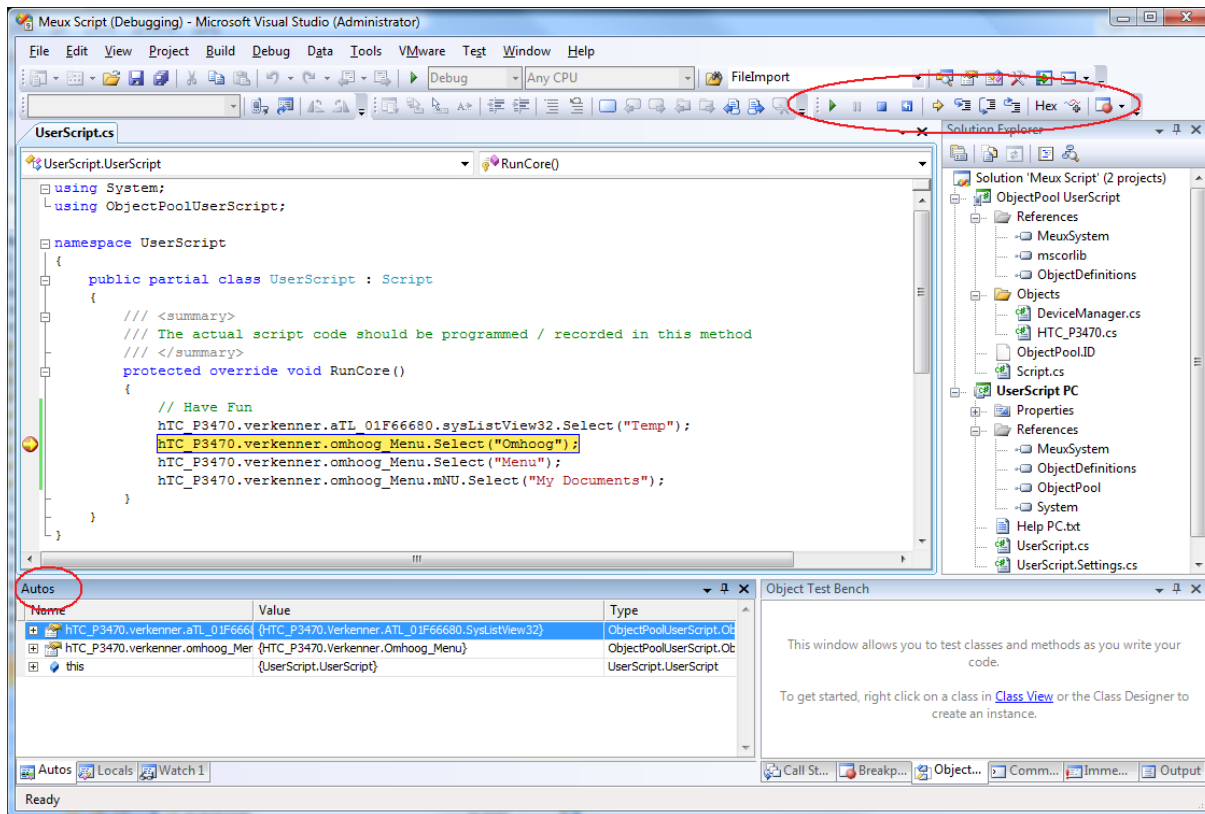
When a whole solution needs to be saved to another directory, the only option is to copy the whole solution folder inside the windows file explorer to another path.

3.9. Replaying a script

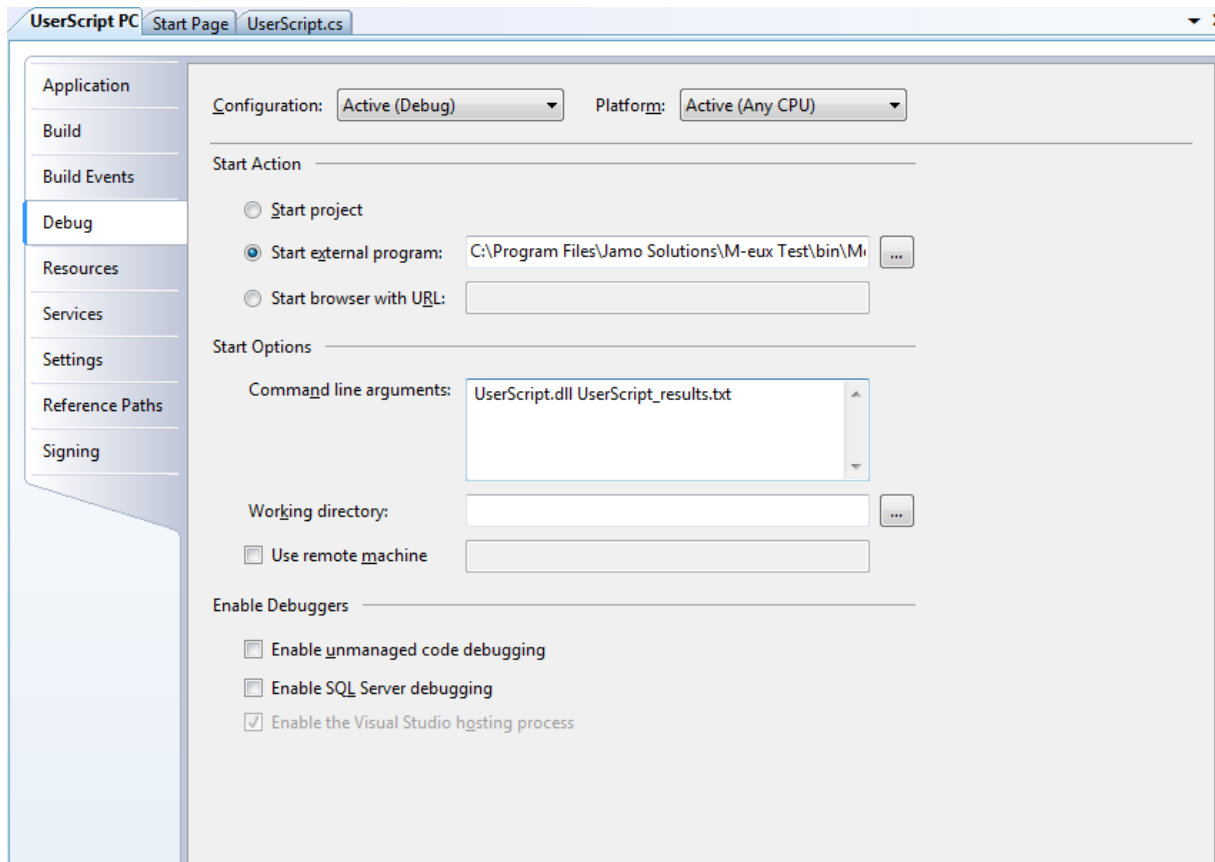
This can be achieved using the normal debugging of visual studio. Breakpoints can be used. Press the green play button inside the visual studio toolbar or go to the debug menu and start debugging. Make sure the tool isn’t recoding anymore.



The debugging features of visual studio will be at disposal.



When a “M-eux Test”-Script is debugged, the application MeuxExecuter.exe is used to run the script (see chapter [Replay a distributed script](#)). These settings can be seen in the project settings window of the userscript project:



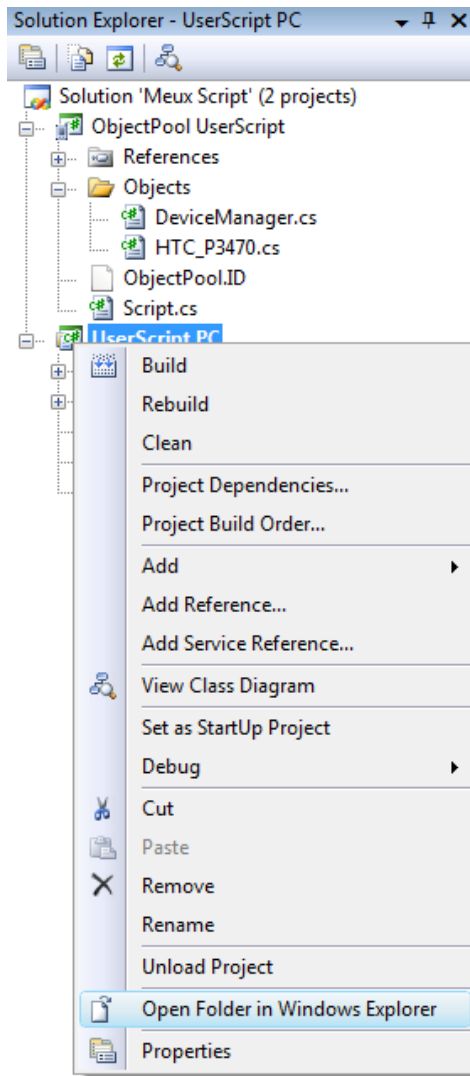
Note: when renaming the script, make sure the “command line arguments”-edit field contains the assembly name of your script.

3.10. Distribute a script

With Visual Studio you can build your script to a .dll-file (default UserScript.dll). You can make a debug-build, or a release-build of the UserScript-project. (see Visual Studio’s Documentation)

Inside the bin-folder of the UserScript project “UserScript.dll” accompanied by “ObjectPool.dll” can be found. These two dll’s belong together and can be distributed to others. (the “.dll”-files discussed above will contain the name specified during the creation of the script.)

Tip: an easy way to navigate directly to the project’s-folder can be found in the right-click menu of the project in the solution explorer of visual studio.



3.11. Replaying a distributed script

1. Make sure a Mobile Device is connected to the M-eux DeviceManager
2. Start “MeuxExecuter” on the pc (start → programs → Jamo Solutions → M-eux test)
3. Browse to the appropriate UserScript.dll
4. Select a name and location for the Result file.
5. The script will run against the mobile device. When finished the Result file will be shown.

3.12. MeuxExecuter Commandline Options

1. [ScriptLocation] : optional. When omitted a dialog will prompt for the Scrip location.
2. [ResultLocation] : optional. When omitted a dialog will prompt for the Result location.

3. /sr [true] or [false]: optional, Indicates if the results will be shown at the end of the test execution. The default value is true.

Example

```
C:\MyScript.dll C:\MyResult.html /sr false
```

3.13. Batch execution

“Meux Executer” is a command-line tool. To run the MeuxExecuter from the command line, following line can be entered in the command prompt:

```
"C:\Program Files\Jamo Solutions\M-eux test\bin\MeuxExecuter.exe" "c:\userscript.dll" "c:\temp\results.htm"
```

When the <enter>-key is pressed, the script "c:\userscript.dll" gets executed and the results will be put in the file "c:\temp\results.htm".

When the last argument or both command line arguments are missing, an open/save dialog box is shown to ask for the missing command line arguments.

An approach for running multiple scripts one after the other is running them from a “.bat”-file.

Example

To run two scripts one after the other you can do it like this:

1. Create a txt file, and put 2 lines in it:
"C:\Program Files\Jamo Solutions\M-eux test\bin\MeuxExecuter.exe" "C:\temp\Meux Script1\Meux Script1\UserScript\bin\Debug\PC\UserScript.dll" "C:\FirstResult.htm"

"C:\Program Files\Jamo Solutions\M-eux test\bin\MeuxExecuter.exe" "C:\temp\Meux Script2\Meux Script2\UserScript\bin\Debug\PC\UserScript.dll" "C:\SecondResult.htm"

(Both lines are too long to fit on one line in this document)
2. Save the file on your disk.
3. Rename the extension “.txt” to “.bat”
4. Now you can double click the bat-file in order to execute the 2 scripts.

3.14. Results of a script, using the ResultManager

Every time a script stops running, a HTML Result file will be generated. To log statements in the resultfile, each Script contains a reference to a ResultManager object.

3.15. Logging statements

There can be logged 5 kinds of statements:

LogPass, LogFail, LogWarning, LogDone, LogError

Example:

```
/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    ResultManager.LogDone("This is a logstatement");
    ResultManager.LogWarning("Be carefull!");
}
```

3.16. GlobalSuccess Property

The ResultManager holds a C#-property GlobalSuccess which holds the error status for the whole script. It indicates whether there already occurred an error or not.

3.17. LogRunStatements Property

The ResultManager also logs each executed Run statement against a ScriptObject. This feature can be enabled or disabled by setting the LogRunStatements property.

```
ResultManager.LogRunStatements = false;
```

It is possible to write your own ResultManager, for more information contact support@jamosolutions.com.

3.18. Exceptions

In each managed .Net application, there is a mechanism for reporting errors, failures and other abnormalities during execution. This mechanism is called “Exceptions”. For example NullReferenceException and DivideByZeroException. See Microsoft’s documentation for more information.

All unhandled Exceptions will cause the script to stop executing immediately. In the result file, the details of the exception will be visible.

If an exception may not interrupt the further execution of a script, try intercepting the exception by using “try-catch”-statements inside the userscript code.

3.18.1. MeuxException

All exceptions occurring during the execution of the script and are somehow related to the “M-eux Test”-product, will manifest as being MeuxExceptions.

The MeuxException class directly inherits from Exception class. There are 4 subclasses of the MeuxException class:

4. MeuxCommunicationException:

Will be thrown when the script cannot communicate with the "M-eux Test"-DeviceManager

5. MeuxObjectNotFoundException

Will be thrown when a ScriptObject cannot be found.

6. MeuxPropertyNotFoundException

Will be thrown when a specified property cannot be found. (getTOPProperty and setTOPProperty methods)

7. MeuxRunException

Will be thrown when a specific method on a ScriptObject fails.

When intercepting a MeuxException, the corresponding ScriptObject, and the name of the method that was being executed, can be retrieved.

Example:

```

/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    bool success = false;
    while (! success)
    {
        try
        {
            // try selecting the temp folder in the file explorer
            hTC_P3470.file_Explorer.atl_01F66680.sysListView32.Select("Temp");
            success = true;
        }
        catch (MeuxSystem.Exceptions.MeuxObjectNotFoundException ee)
        {
            // log the error
            ResultManager.LogError(ee);
            // launch the file explorer on the mobile device
            hTC_P3470.AppLaunch("fexplore", "", "");
        }
        catch (MeuxSystem.Exceptions.MeuxRunException ee)
        {
            // log the error
            ResultManager.LogError(ee);
            // select the first button on the menu bar to go up one directory
            hTC_P3470.file_Explorer.omhoog_Menu.Select("#0");
        }
    }
    hTC_P3470.file_Explorer.Close();
}

```

3.19. Run one script from another Script

One way for executing multiple scripts is by **executing them in a ".bat"-file**.

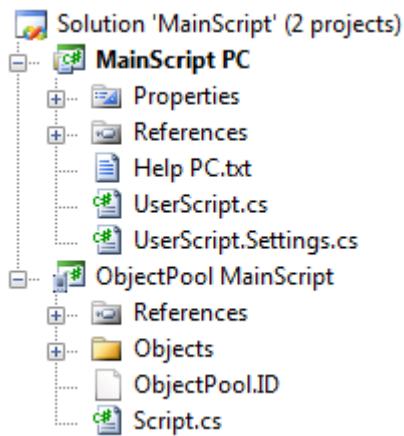
A more advanced approach is invoking one script from another. Playing around with visual studio projects and references gives a lot of freedom for achieving such functionality.

What follows is a guideline example of how this can be achieved.

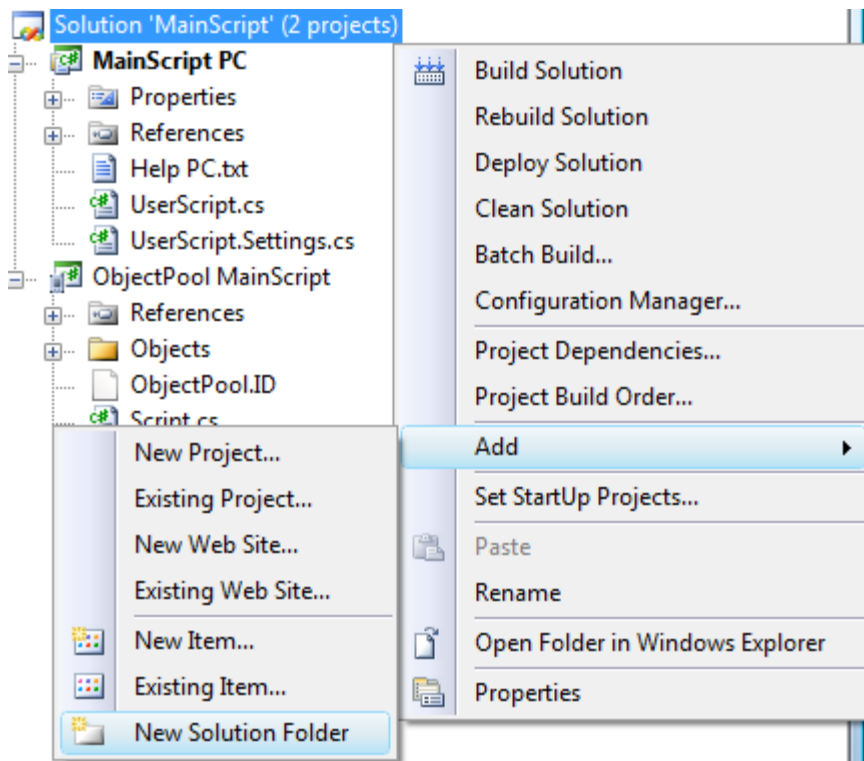
Example

This example will create a main script that will invoke two other scripts.

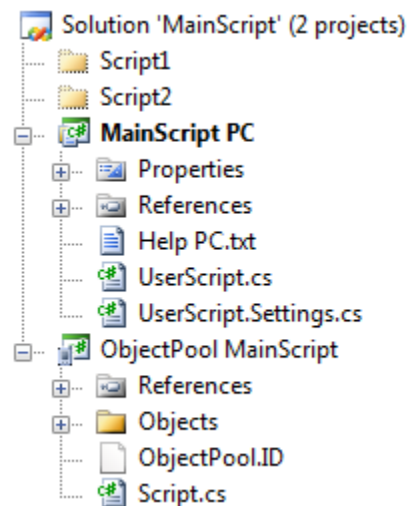
1. Create a new script, name it "Script1". (File → New Project → "M-eux Test"-Script)
2. Build the Solution. (Right-click the solution in the solution explorer → Build Solution)
3. Close the Solution (File → Close Solution).
4. Create a new script, name it "Script2".
5. Build the Solution.
6. Close the Solution.
7. Create a new script, name it "MainScript".



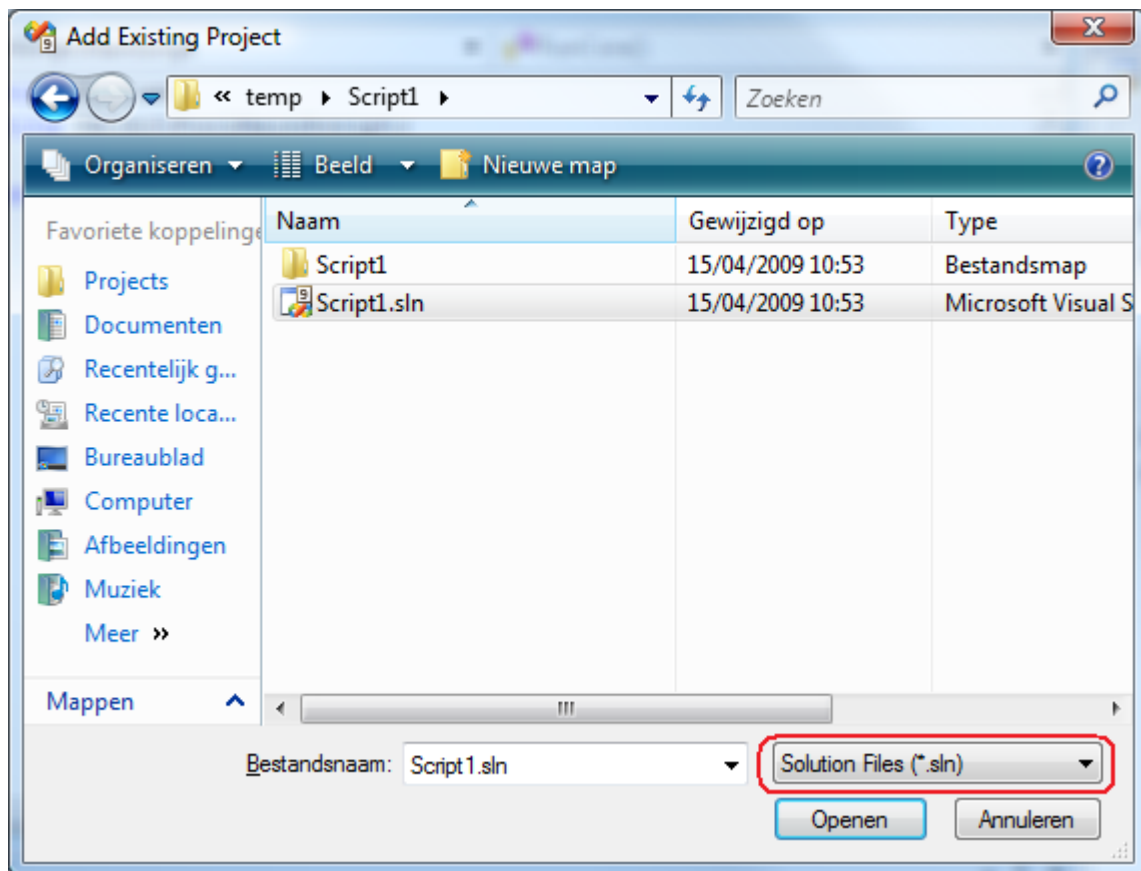
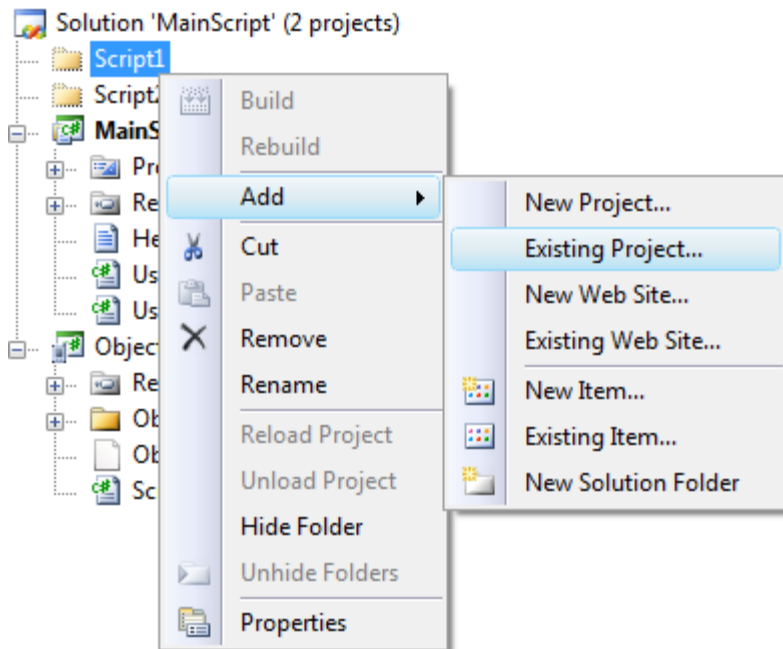
8. The other scripts need to be known inside the MainScript, therefore some references need to be added, before this is done, the solution should be well organized to keep an overview all the time.
9. Add a new solution folder and name it "Script1", this is only for keeping the solution more user friendly.



10. Add a new solution folder and name it “Script2”.

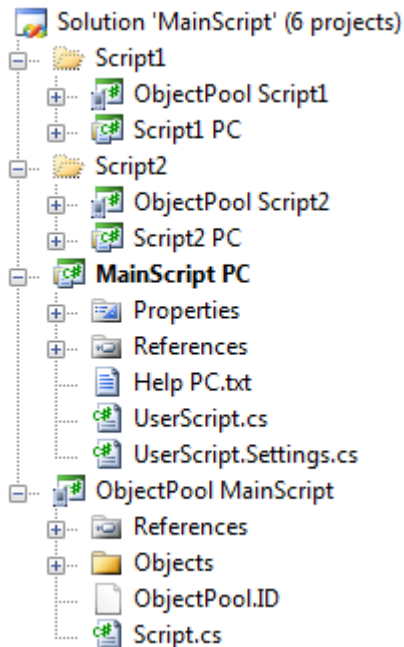


11. Add the solution of the first script to the “Script1”-solution folder.

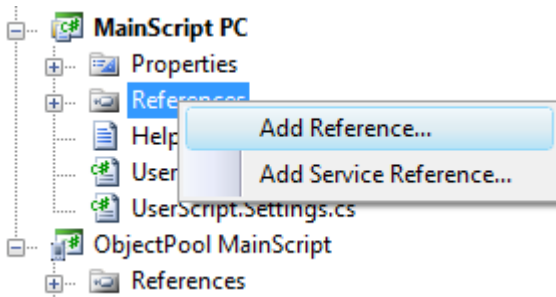


12. Add the solution of the second script to the "Script2" - solution folder.

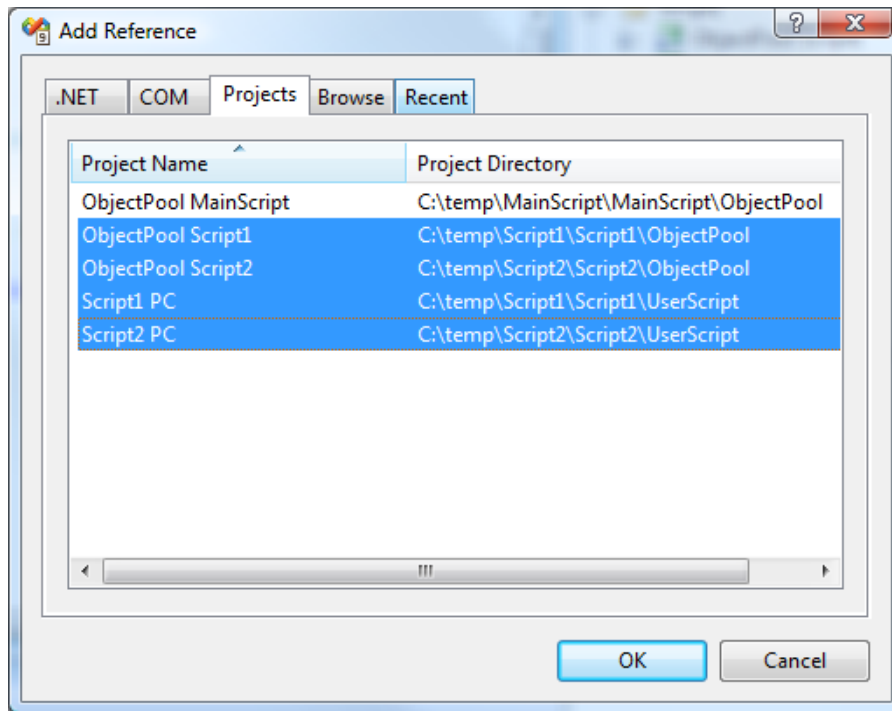
13. When adding a solution to another solution, all projects inside the solution are added. Projects are added by reference, there is no copy taken.



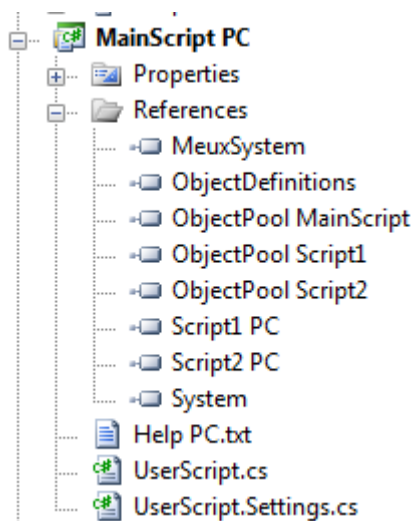
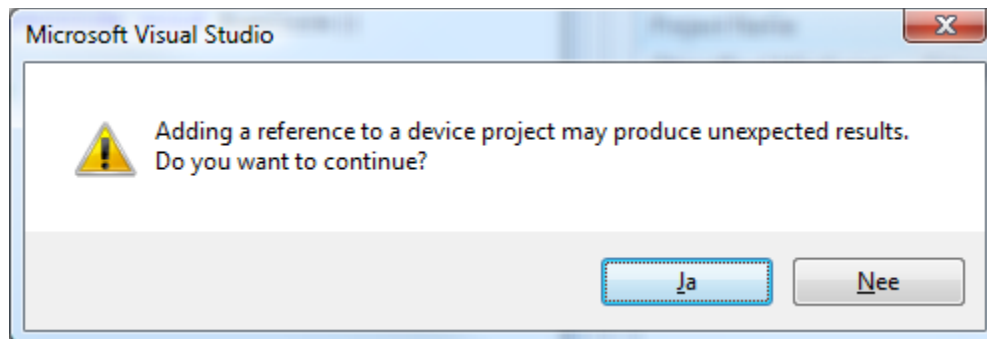
14. Add the Script1 and Script2 projects as references in the MainScript Project.



Multiselect is possible:



15. If warnings are shown, click yes.



16. Implement following code in mainscript.cs:

```
using System;
using ObjectPoolMainScript;

namespace MainScript
{
    public partial class MainScript : Script
    {
        /// <summary>
        /// The actual script code should be programmed / recorded in this method
        /// </summary>
        protected override void RunCore()
        {
            // set the resultmanager for script1
            Script1.Script1.Instance.ResultManager = ResultManager;
            // run script1
            Script1.Script1.Instance.Run();

            // set the resultmanager for script2 and run the script
            Script2.Script2.Instance.ResultManager = ResultManager;
            // run script2
            Script2.Script2.Instance.Run();
        }
    }
}
```

17. Build the solution.

The MainScript Solution contains 6 projects. No Record and “Learn Gui” are supported for this solution. Debugging is possible.

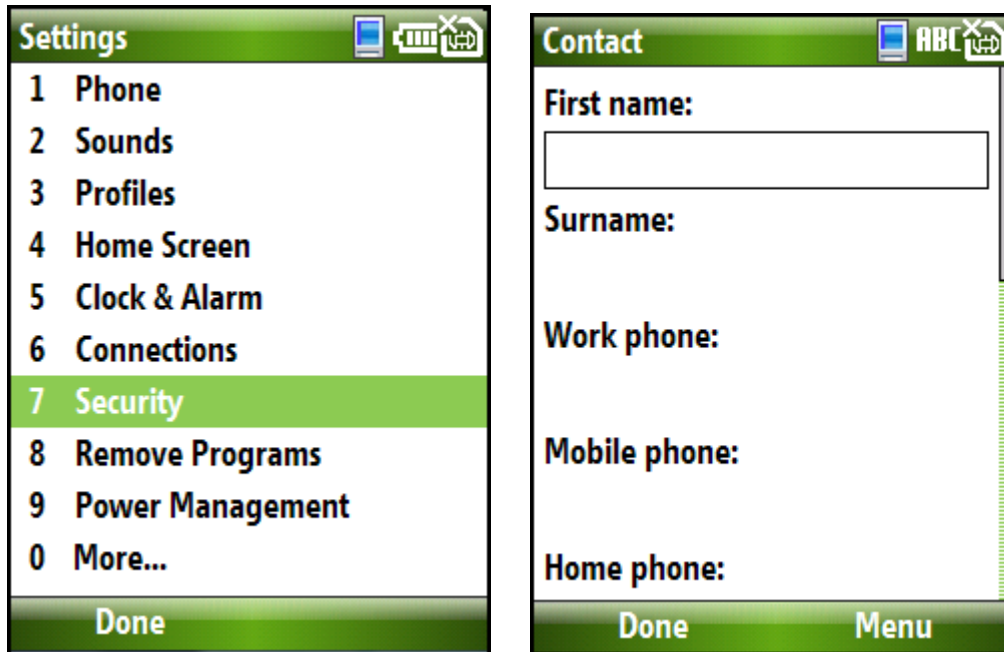
The output folder of the mainscript project will contain all script dll’s and all objectpool dll’s.

Record and “Learn Gui” can still be done in the original solution of Script1 and the original solution of Script2. The changes will be automatically reflected in the MainScript.

Chapter 4: Advanced Options

4.1. The MoListViewCE object in Windows Mobile Standard Edition

The MoListViewCE object is an object specific for the Windows Mobile Standard platform. The object is used to put strings or other GUI objects in a one-column layout format. Examples are:



On the left is the Settings screen and on the right the screen to add a new contact. Both are structured by the MoListViewCE object. The MoListViewCE object does represent an object for which Microsoft did not release a public interface. The object is supported in analog mode. No text, no items or item count can be retrieved from the object.

One method is supported on the object: the Press method for replaying the press of a key against the object.

Following script shows the navigation down the list and the selection of an item using the fast key "7":

```
/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    smartPhone.settings.mS_LISTUI_CE_1_0.mS_VIRTUAL_LIST_VIEW_CE_1_0.Press("VK_TDOWN");
    smartPhone.settings.mS_LISTUI_CE_1_0.mS_VIRTUAL_LIST_VIEW_CE_1_0.Press("VK_TDOWN");
    smartPhone.settings.mS_LISTUI_CE_1_0.mS_VIRTUAL_LIST_VIEW_CE_1_0.Press("VK_TDOWN");
    smartPhone.settings.mS_LISTUI_CE_1_0.mS_VIRTUAL_LIST_VIEW_CE_1_0.Press("VK_TDOWN");
    smartPhone.settings.mS_LISTUI_CE_1_0.mS_VIRTUAL_LIST_VIEW_CE_1_0.Press("7");
}
```

When you have a MoListViewCE with other GUI elements than text, then the down or up navigation is not against the list but against the opened GUI element. Following scripts show the fill in of the fields of a new contact record:

```

/// <summary>
/// The actual script code should be programmed / recorded in this method
/// </summary>
protected override void RunCore()
{
    smartPhone.contacts.new_Menu.Select("New");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.cAPEDIT.Set("Jacques");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.cAPEDIT.Press("VK_TDOWN");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.cAPEDIT.Set("Mda");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.cAPEDIT.Press("VK_TDOWN");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.edit.Set("016785236");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.edit.Press("VK_TDOWN");
    smartPhone.contact.ms_LISTUI_CE_1_0.ms_VIRTUAL_LIST_VIEW_CE_1_0.edit.Set("047523652");
    smartPhone.contact.done_Menu.Select("Done");
}

```

4.2. Testing .Net Compact Framework Applications

This section explains how .Net Compact Framework Applications can be tested in an extended way. The GUI controls of a .Net application can be recognized by the standard functionality of M-eux Test. The product M-eux Test provides however extended support for the .Net controls. This section and its sub sections describe how this extended support can be activated and the functionality provided in this extended support.

Both the .Net Compact Framework 2.0 and 3.5 are supported.

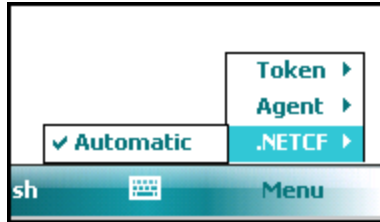
4.2.1. Automatic activation of the .Net Compact Framework Extended support.

The automatic activation of the extended support can be used for .Net Framework applications 1.0, 2.0 and 3.5. The extended support is however not activated by default on the device.

4.2.2. How to active the automatic .Net extended support?

You have to open the 'M-eux Control Panel' applications. If this application shows that the agent is running, then you have to stop the agent. You can stop the agent by selecting the menu 'Menu' on the menu bar, sub menu agent and from this sub menu the entry labeled 'Stop'.

Then you activate on the device the automatic .Net extended support by selecting the menu 'Menu' on the menu bar, sub menu '.NETCF' and from this sub menu check the entry labeled 'Automatic'. The automatic extended is active on the device, if this menu entry is checked as illustrated in following figure:



Restart the agent. The enablement is saved in the registry of the device. If you are working in a windows mobile emulator and you do a hard reset or if you reset the real device to the factory default settings, then you have to re-install the agent and you have to re-activate the Automatic extended support for .Net CF applications.

In the .Net CF application under test, the extended support is only enabled when opening the application from the script. You have to launch the application using the '.NetCFAppLaunch' or '.NetCDAppLaunch35' commands of the MobileDevice test object. If the application is not opened using this command, then the extended support is not enabled. After launching the application with the '.NetCFAppLaunch' or '.NetCFAppLaunch35' command, the tester can start creating his script by recording, by learning the GUI or by other script actions. Also for replaying the script, the .Net CF application under test needs to be launched by the .NetCFAppLaunch or .NetCFAppLaunch35 commands.

The command .NetCFAppLaunch35 is used for .Net Compact Framework 3.5 applications. The command .NetCFAppLaunch is used for .Net Compact Framework 2.0 and 1.0 applications.

When the .Net CF application is launched, the application can be closed in the normal way. After closing the application, the .NetCFAppStop command needs to be executed. This command will clean up the .Net CF extended support.

A script should need to look like:

```
'Launch the .Net CF application

Id = MobileDevice("X1i").NetCFAppLaunch "MoneyGoesAway.exe", "\Program Files\MoneyGoesAway", ""

' Perform different actions

' after closing the application or to close the application

MobileDevice("WindowsCE").NetCFAppStop Id
```

4.2.3. Enabling extended support in a non-automatic way

In order to use the extended support whereby this support is not enabled automatically, a minor source code modification is required. Note that this code change is required for .Net CF 3.5. There are two approaches:

- “GetForegroundControl”-approach. In this approach, the application needs also be launched with the “NetCFAppLaunch” command.
- “Register Form”-approach

In the installation folder on the pc there is a sample available of the .Net CF extended support. This sample includes both a visual studio solution of a demo application and a Visual Studio script. You must build the visual studio solution and put the resulting exe file on the device in order to run the script.

GetForegroundControl-approach

To make a .Net application testable, the developer has to add following method in the code with the signature:

```
private static Control GetForegroundControl()  
{  
    ...  
    return ...;  
}
```

Each time this method is called, it should return the current foreground Control, mostly this will be a Form.

When the .Net CF source code modification is made, the extended .Net support is only accessible using the NetCFAppLaunch method in the scripting environment to startup the application.

NetCFAppLaunch can only be used when the source code modification is implemented.

Example:

```
hTC_P3470.NetCFAppLaunch("MyNetCFApplication.exe", "", "");
```

For more information see the reference guide.

Example

When a new .Net compact framework project is generated, following code will be present:

```
static class Program  
{  
    /// <summary>  
    /// The main entry point for the application.  
    /// </summary>  
    [MTAThread]
```



```
static void Main()
{
    Application.Run(new Form1());
}
```

The Application is made testable as follows:

```
static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [MTAThread]
    static void Main()
    {
        form = new Form1();
        Application.Run(form);
    }

    private static Form form;

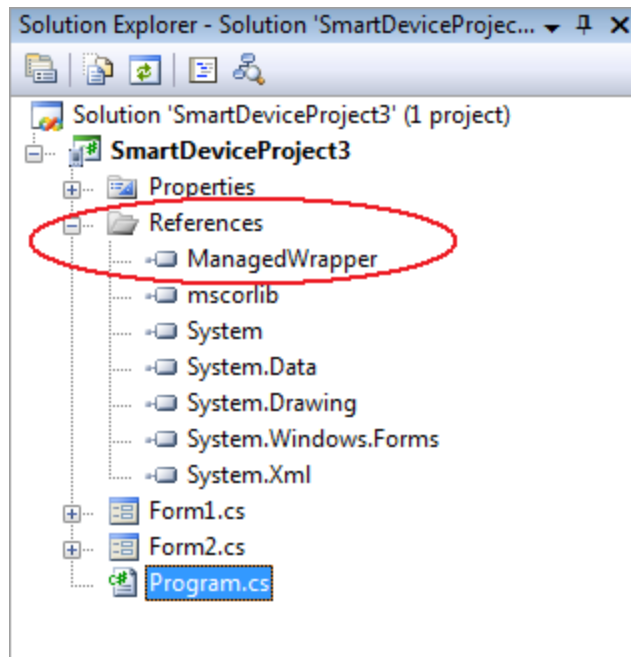
    private static Control GetForegroundControl()
    {
        return form;
    }
}
```

Note: this example assumes only one form exists in the application, it will always be the foreground control.

4.2.4. Register Form-approach

In the development environment of the .Net CF Application:

Add a reference to the file “ManagedWrapper.exe”. This file can be found in the lib directory of the installation folder of M-eux Test.



- Whenever a form is created, invoke the “Register”-method inside “managedWrapper.exe” and pass a reference to the form. For example:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        ManagedWrapper.Program.Register(this);
    }
    ...
}
```

4.2.5. Overview extended functionality

The extended functionality is explained by using the sample application available in the installation directory of M-eux Test. This sample application used the GetForegroundControl adaptation.

Please execute following step in order to use the sample application:

1. Copy the executable to the windows folder (for simplicity) on the device.
2. Set-up the testing environment (device manager / Visual Studio / agent)
3. Generate a new Visual Studio script, and put following statement:
4. `hTC_P3470.NetCFAppLaunch("demoNetCF.exe", "", "");`

Note: the NetCFAppLaunch is only needed when the “GetForegroundControl”-Approach is applied.

5. Start recording. Tap on the button, tap on the calendar object.

The recorded script should look like:

“GetForegroundControl”-Approach:

```
hTC_P3470.NetCFAppLaunch("demoNetCF.exe", "", "");  
hTC_P3470.formDemo.buttonDemo.Push();  
hTC_P3470.formDemo.monthCalendarDemo.Tap ("55","82");  
hTC_P3470.formDemo.monthCalendarDemo.Tap ("139","15");
```

“Register Form”-Approach:

```
hTC_P3470.formDemo.buttonDemo.Push();  
hTC_P3470.formDemo.monthCalendarDemo.Tap ("55","82");  
hTC_P3470.formDemo.monthCalendarDemo.Tap ("139","15");
```

In the ObjectPool you can see that all objects are categorized in their MS .Net CF class, and an extra identification property “Name” is used.

In the script, the tester can call extra functions to cope with controls’ managed properties and methods. The sample Visual Studio script in the “M-eux test”-installation directory shows how you can call managed properties and methods by programming the script.

The previous script can be modified like this:

“GetForegroundControl”-Approach:

```
hTC_P3470.NetCFAppLaunch("demoNetCF.exe", "", "");  
hTC_P3470.formDemo.buttonDemo.Push();  
hTC_P3470.formDemo.monthCalendarDemo.SetNetCFROProperty("TodayDate", "20-11-2008");  
hTC_P3470.formDemo.monthCalendarDemo.SetNetCFROProperty("SelectionStart", "8-11-2008");
```

“Register Form”-Approach:

```
hTC_P3470.formDemo.buttonDemo.Push();  
hTC_P3470.formDemo.monthCalendarDemo.SetNetCFROProperty("TodayDate", "20-11-2008");  
hTC_P3470.formDemo.monthCalendarDemo.SetNetCFROProperty("SelectionStart", "8-11-2008");
```

The SetNetCFROProperty sets a specific property of the .Net object implementing the calendar object.

Executing methods

Executing a .Net CF method in the Visual Studio scripting environment can be done as follows:

```
hTC_P3470.formDemo.buttonDemo.ExecuteNetCFMethod("Hide");
```

It is also possible to execute custom methods on custom controls. Click [here](#) for more information.

4.2.6. Getting and Setting Properties

An example how a script can take care of .Net CF properties:

```
//Get the Width Property of a button  
string width = hTC_P3470.formDemo.buttonDemo.GetNetCFROProperty("Width");  
  
//Set the Height Property of a button to 50  
hTC_P3470.formDemo.buttonDemo.SetNetCFROProperty("Height", "50");
```

The ReferenceGuide explains these methods.

It is also possible to handle custom properties on custom controls. Click [here](#) for more information.

4.2.7. Parameters and return values

All parameters of a method in the M-eux-script (or the value of a setter) are passed as a string, if a compact framework method requires for example an integer parameter, the parameter will be parsed by the agent on the device.

Only parameter types which can be formed by parsing a string are supported. E.g. Boolean, Int32, DateTime all have a static method Parse:

Int32:

```
public static int Parse(string s);
```

Boolean:

```
public static bool Parse(string s);
```

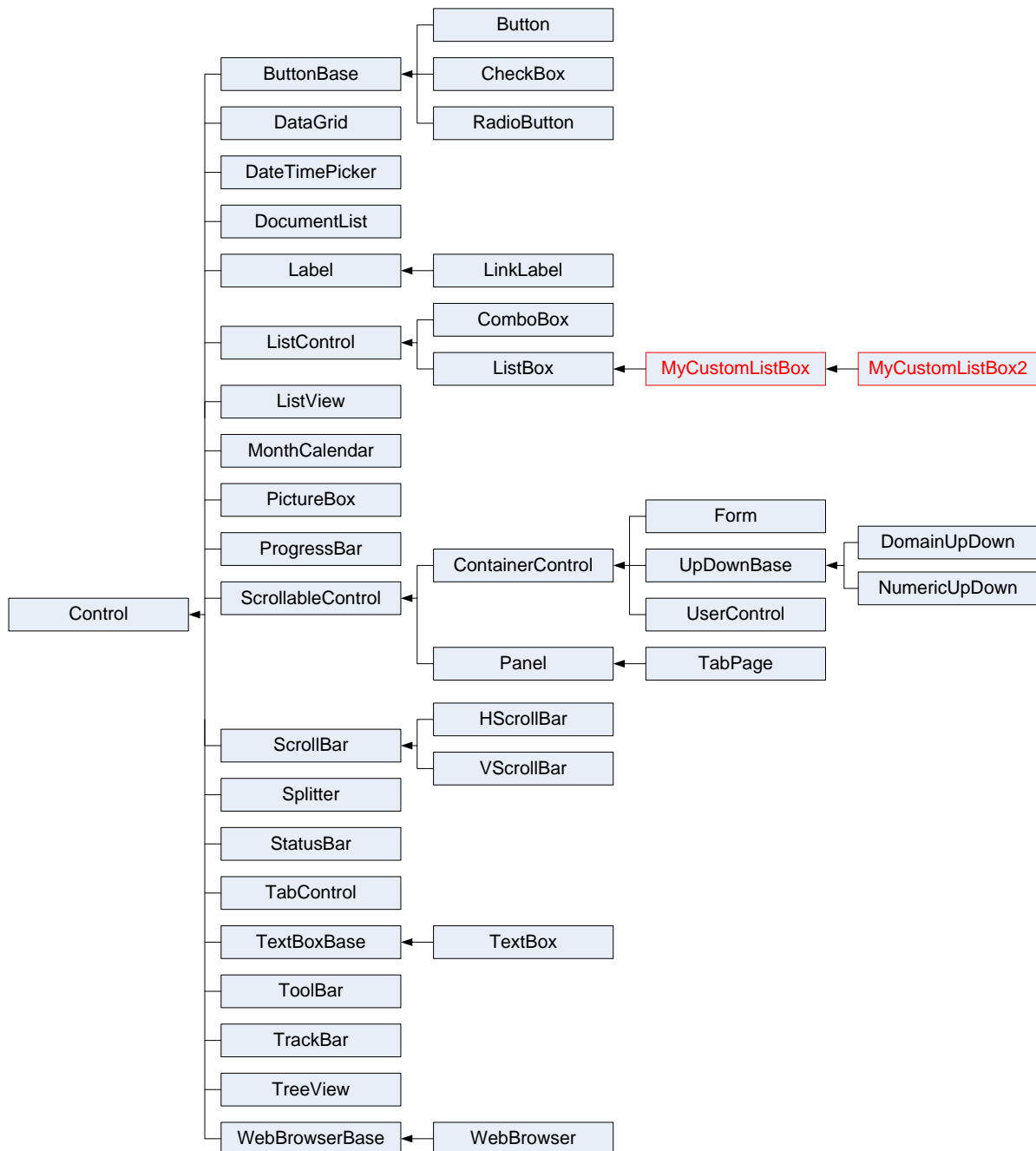
DateTime:

```
public static DateTime Parse(string s);
```

If a method on a control in your application requires a parameter with type a customized class, provide that customized class with a static Parse method like above and you can support that parameter in the Visual Studio scripts. The same parsing logic applies for setting properties.

When a method returns a value, or a property is retrieved, it will be returned as a string. All .Net Compact Framework classes support the ToString method which will be called to return values to the M-eux script in Visual Studio.

4.2.8. Supported controls



The overview above is a simplified class diagram of all default Controls available in the .Net Compact Framework. The controls marked in red are customized controls.

All default controls (black) are known by the M-eux objectPool inside Visual Studio, the test object's .Net-class will be "MoNetCF" plus the Name of the class in the .Net Compact Framework. For example a .Net Compact Framework Button will appear in the ObjectPool as being a "MoNetCFButton".

All .Net Compact Framework test objects are extensions of regular Windows MFC test objects; as a consequence all identification properties and available methods are inherited, i.e. a MoNetCFButton is an extension of the MoButton test object. A very important identification property that becomes available for .Net Compact Framework test objects is the Name Property of the corresponding control.

Custom controls are also supported in the M-eux test solution. They will not have their own test object's class, but will appear in the ObjectPool as a known parent class.

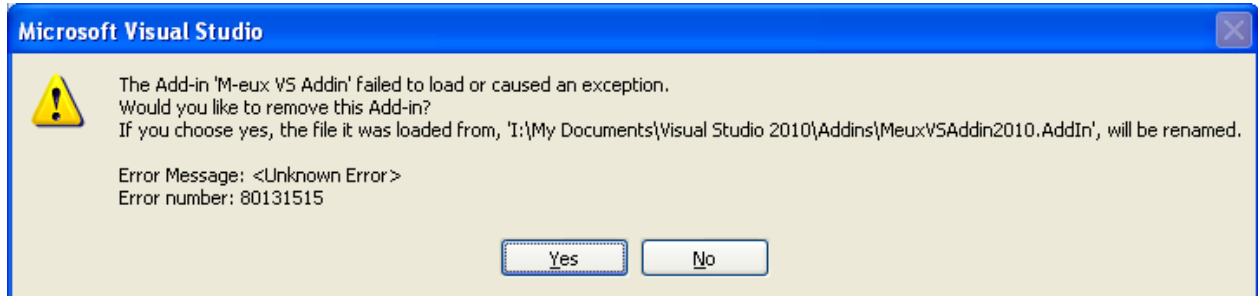
When an unknown control is used in the application, the m-eux agent on the device will look up the .Net Compact Framework inheritance tree to find the first default control.

In the class diagram there are two examples of customized controls MyCustomListBox and MyCustomListBox2. They will both being recognized as a MoNetCFListBox.

However, this will not be a constraint in any way, since all customized methods and all custom defined properties on these controls are accessible.

Chapter 5: Troubleshooting

5.1. The Add-in Meux VS Addin failed to load in VS 2010.



If you get an exception like this please select “No”. This error is caused by one of our assemblies not running with full trust. By adding the following snippet to devenv.exe.config (under "C:\Program Files (x86)\Microsoft Visual Studio X\Common7\IDE" directory) as specified in the msdn documentation the problem should be resolved:

```
<configuration>

...

<runtime>

...

<loadFromRemoteSources enabled="true"/>

...

</runtime>

...

</configuration>
```

Chapter 6: Using M-eux Test with Unit Test Projects

By default, M-eux Test uses its own test projects and testing framework. If you want, you can write your mobile tests as unit tests within Visual Studio. This allows you to:

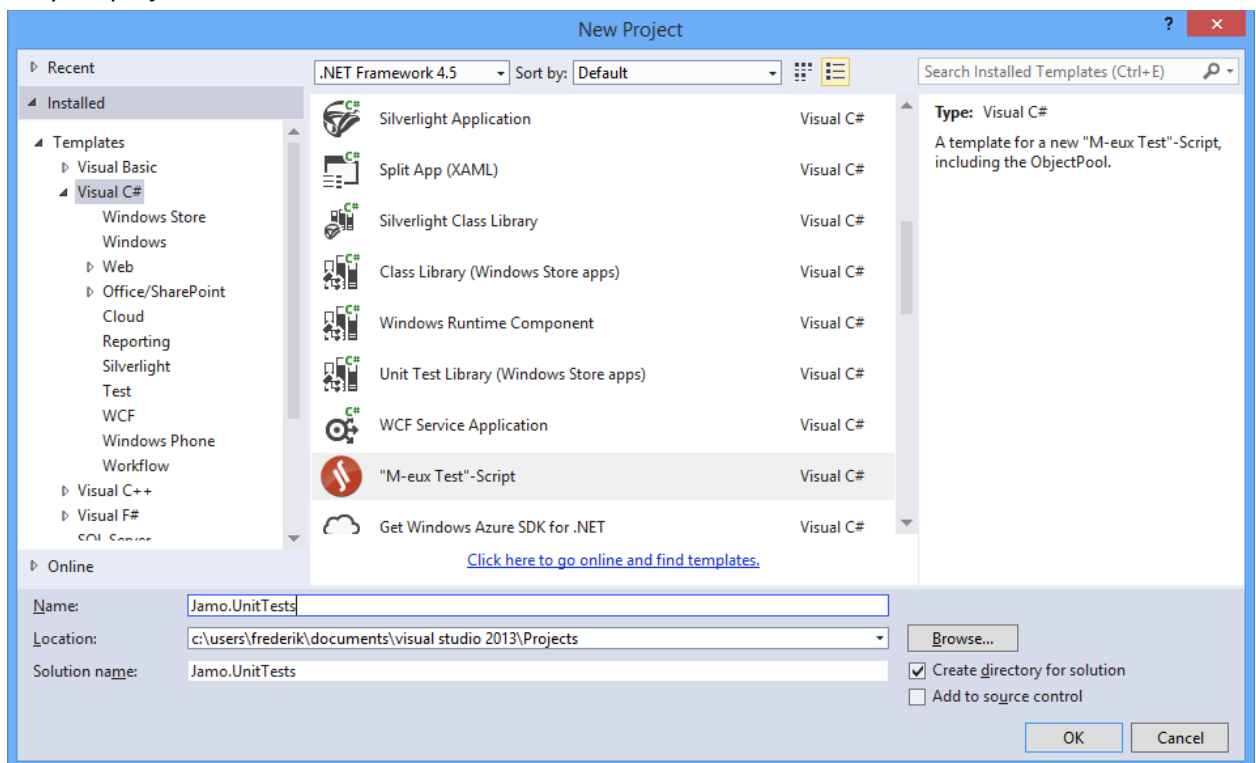
- Use M-eux Test and the unit testing framework from Visual Studio for your mobile tests;
- Automate your test cases in Microsoft Test Manager (MTM) using M-eux Test;
- Use M-eux Test with the Load and Performance Testing features of Visual Studio;
- Integrate with the Team Build features of Team Foundation Server;

6.1. Create your Unit Test Project

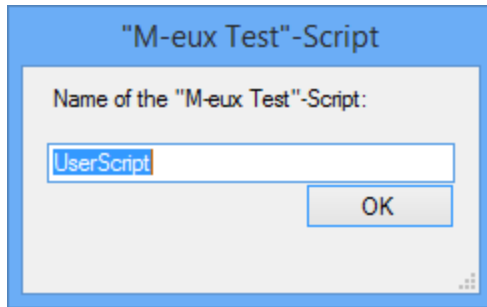
To create a new Unit Test project in Visual Studio that is compatible with M-eux Test, you will first need to create a M-eux Test solution. You can then add a Unit Test project to your solution.

To do so, follow these steps:

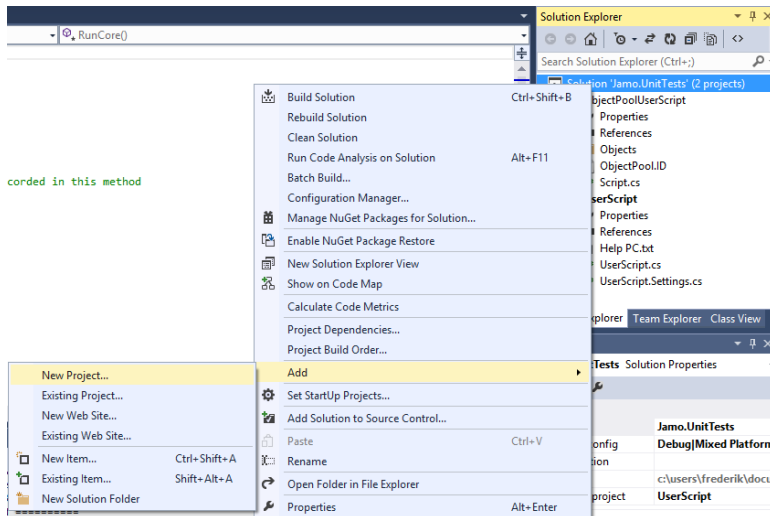
1. In Visual Studio, click **File, New** and choose **Project**
2. In the **New Project** window, select **"M-eux Test" -Script** in the project window. Provide a name for your project and solution and click **OK**.



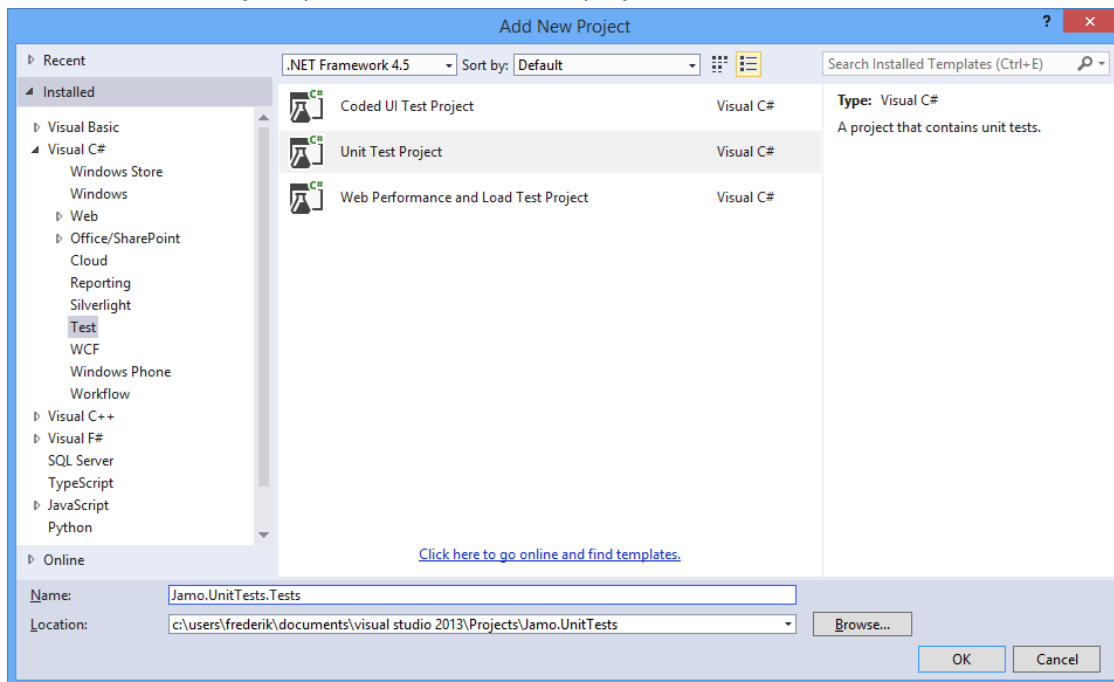
3. In the **"M-eux Test" -Script** dialog box, type the name of the script and click OK. You can accept the default value.



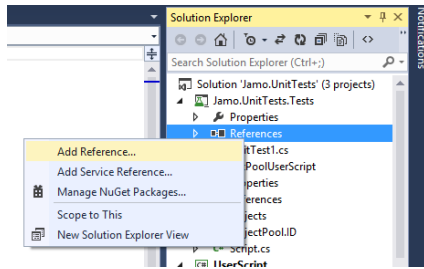
4. In both projects, you can remove the reference to Microsoft.CSharp.
5. Delete the **UserScript** project.
6. In the **Solution Explorer** window, right-click the solution and choose **Add, New Project...**



7. Select **Unit Test Project**, provide a name for the project and click **OK**



8. Right-click the **References** node in the Test Project and select **Add Reference**.

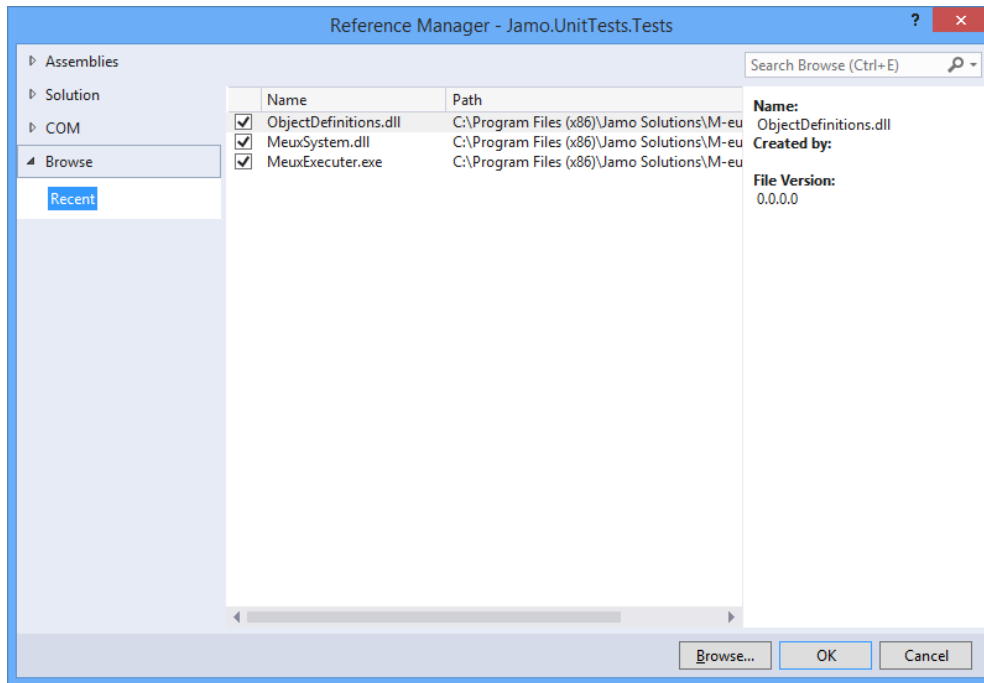


9. Check the **ObjectPoolUserScript** project.

10. Then, click **Browse** and add a reference to the following files:

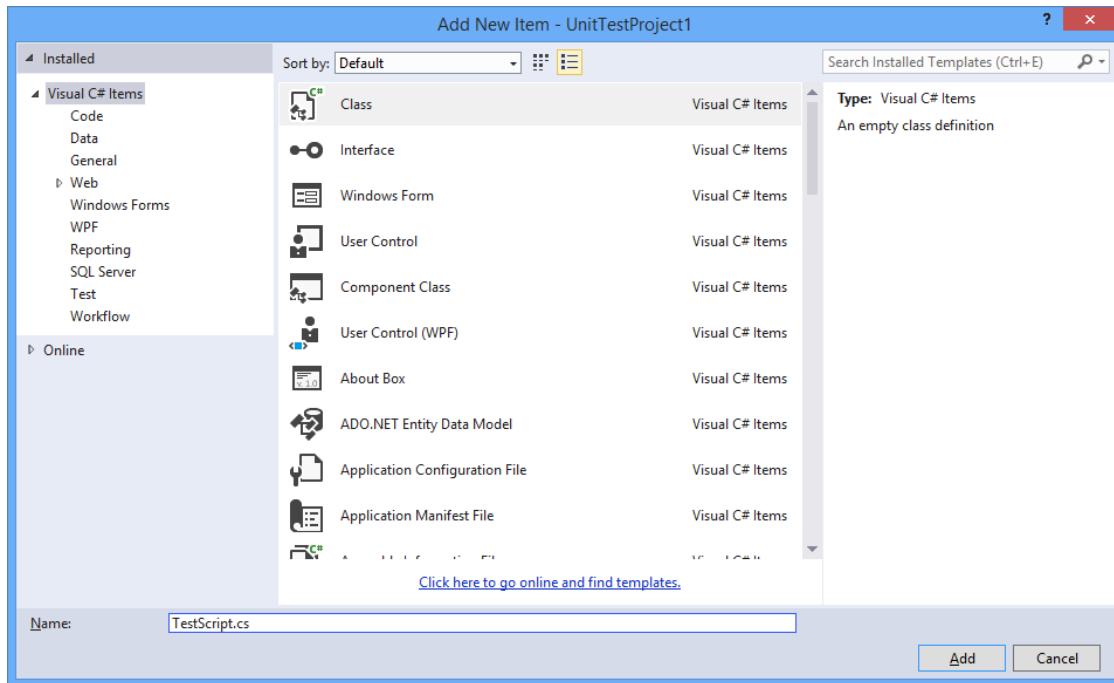
1. C:\Program Files (x86)\Jamo Solutions\M-eux Test\bin\MeuxExecuter.exe
2. C:\Program Files (x86)\Jamo Solutions\M-eux Test\bin\MeuxSystem.dll
3. C:\Program Files (x86)\Jamo Solutions\M-eux Test\bin\ObjectDefinitions.dll

Then, click **OK**.



11. Right-click the project and click **Add, Class...**

12. Provide the name **TestScript** and click **OK**



13. At the top of the file, replace the using declarations with the following code:

```
using MeuxExecuter;
using MeuxSystem;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using ObjectPoolUserScript;
using System;
```

14. Within the namespace { } declaration, change the class definition to the following:

```
/// <summary>
/// The <see cref="ScriptExecution"/> class manages the connection between the
testing framework
/// and the M-eux Test Device Manager.
/// </summary>
private ScriptExecution scriptExecution;

/// <summary>
/// Initializes a new instance of the <see cref="TestScript"/> class.
/// </summary>
public TestScript()
: base(new MeuxReplaySettings())
{
    this.MeuxReplaySettingsFindObjectTimeOut = 20000;
}

/// <summary>
/// Gets or sets the test context which provides
information about and functionality for the current test run.
///</summary>
public TestContext TestContext
{
    get;
    set;
}
```

```

}

/// <summary>
/// Dummy implementation of the base <see cref="Script.RunCore"/> method.
/// </summary>
[Obsolete]
protected override void RunCore()
{
}

/// <summary>
/// Initializes the connection between the Unit Testing Framework and
/// the M-eux Test Device Manager.
/// </summary>
[TestInitialize]
public void InitializeConnection()
{
    scriptExecution = new ScriptExecution();
    scriptExecution.OpenRunEnvironment();
}

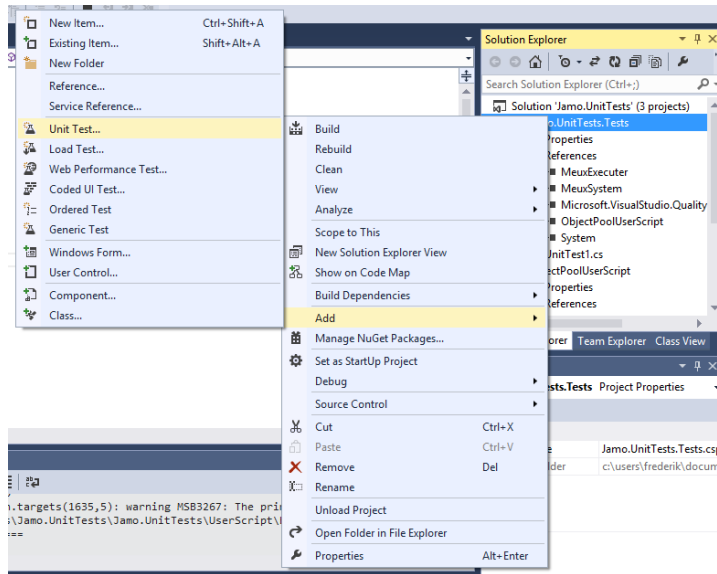
/// <summary>
/// Tears down the connection between the Unit Testing Framework and
/// the M-eux Test Device Manager.
/// </summary>
[TestCleanup]
public void CloseConnection()
{
    scriptExecution.CloseRunEnvironment();
}

```

6.2. Create your unit test class

Unit tests that target the UI of your mobile applications need to follow a specific structure. In this section, we guide you through the process of creating a new Unit Test that can automate the UI of your mobile applications.

1. Right-click the test project and select **Add, Unit Test...**



2. Add the top of the file, add the following code:

```
using MeuxSystem;
using MeuxExecuter;
using ObjectPoolUserScript.Objects;
using ObjectPoolUserScript;
```

3. Change the declaration of your Unit Test class so that it inherits from the TestScript class. For example:

```
public class UnitTest2 : TestScript
```

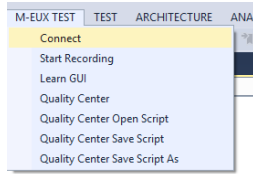
6.3. Record your first Unit Test

You can now record your first unit test:

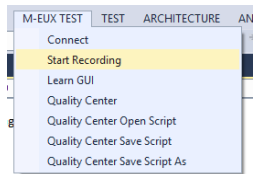
1. Make sure:
 1. The **Device Manager** is started
 2. At least one device is connected to the Device Manager
 3. You have a testable application running on your device.



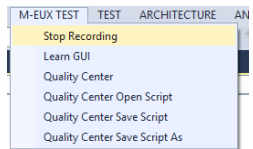
2. In Visual Studio, click **M-eux Test** and **Connect**



3. Place the cursor in the TestMethod1() method in your Unit Test class.
4. In Visual Studio, click **M-eux Test** and **Start Recording**.



5. Execute a couple of actions on your mobile device. They will be recorded as a test script.
6. When you are done, click **M-eux Test** and **Stop Recording**.



7. You may need to repeat steps **Error! Reference source not found.** through **Error! Reference source not found.** from section 6.1.
8. You can add validation logic to your test script by using the standard Visual Studio Unit Testing features, such as the Assertion class. For more information, see [Using the Assert Classes](#) and [Writing Unit Tests for the .NET Framework with the Microsoft Unit Test Framework for Managed Code](#) on MSDN.

6.4. Converting a Unit Test into a Performance Test

You can use M-eux Test to test the performance of your mobile application in two ways:

- You can measure the response times of the UI of your mobile applications
- You can automate your application on multiple devices at the same time to generate load against your back-end servers and measure e.g. how well the servers behave when multiple users access them concurrently.

To support these test cases, we need to update the base TestScript class or the individual unit test classes so that they can accommodate for the following:

- Measure execution times and response times on the devices.
- Handle concurrent execution on multiple devices correctly. We implement this by tying each VUser to a specific device. Each VUser has a unique ID. We suggest that you also name the devices so that their name ends with a unique ID (e.g. iPhone01, iPhone02,...). This will make it easy for you to link VUsers to devices.

You can easily convert an existing unit test into a performance test. To do so, take the following steps:

1. Add the following field declarations to your TestScript class:

```
private bool inLoadTest;
private int  userId;
private string  deviceName;
```

2. Add the following method, ExecuteWithTiming, to your test class. It takes an Action delegate and will measure how long it takes for that code to execute:

```
private void ExecuteWithTiming(string name, Action method)
{
    // Find out if we're running in a load test or unit test. If we're in a load
    // test, there will be a
    // property called AgentID
    bool inLoadTest = TestContext.Properties.Contains("AgentId");

    if (inLoadTest)
    {
        TestContext.BeginTimer(name);
    }

    method();

    if (inLoadTest)
    {
        TestContext.EndTimer(name);
    }
}
```

3. At the end of the InitializeConnection() method, add the following code. You need to replace:
 1. android- with the prefixes you use for your devices. For example, if you name your devices iPhone01, iPhone02 and so on, the prefix would be iPhone.
 2. android_01 with the name of your device in the object repository. For example, if you recorded your scripts using a device named iPhone, you would replace android_01 by iPhone.

```
inLoadTest = TestContext.Properties.Contains("AgentId");

if(inLoadTest)
{
    // Bind to the correct virtual device
    LoadTestUserContext loadTestUserContext =
    this.TestContext.Properties["$LoadTestUserContext"] as LoadTestUserContext;
    userId = loadTestUserContext.UserId + 1; // VS user ids are zero-based
    deviceName = string.Format("android-{0:D2}", userId);
    TestContext.WriteLine(string.Format("Executing RunCore() for user with ID {0},
    using device {1}", userId, deviceName));
    Debug.WriteLine(string.Format("Executing RunCore() for user with ID {0}, using
    device {1}", userId, deviceName));
    this.android_01.SetTOProperty("name", deviceName);
}
```

4. In your Unit Test methods, embed the code for which you want to measure the performance in the ExecuteWithTiming() method. For example:

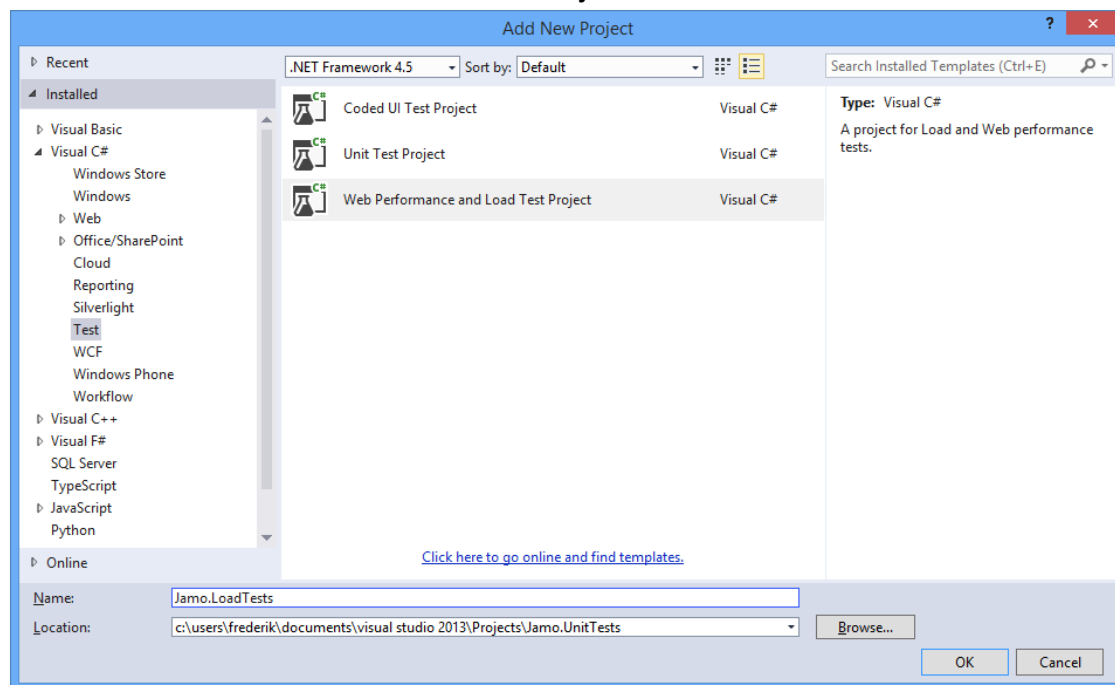
```
// Navigate to the Visibility screen, record timings.
ExecuteWithTiming("Navigate to Visibility Screen",
    delegate()
    {
        apiDemos.Select("Views");

        apiDemos.Select("Visibility");
    });
```

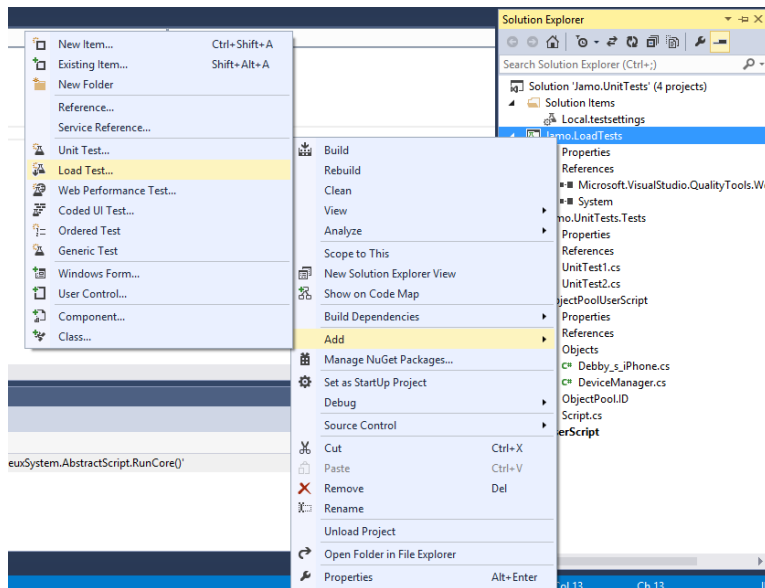
6.5. Adding your unit tests to a performance test project

To create a performance test project and add your unit tests to that project, follow these steps:

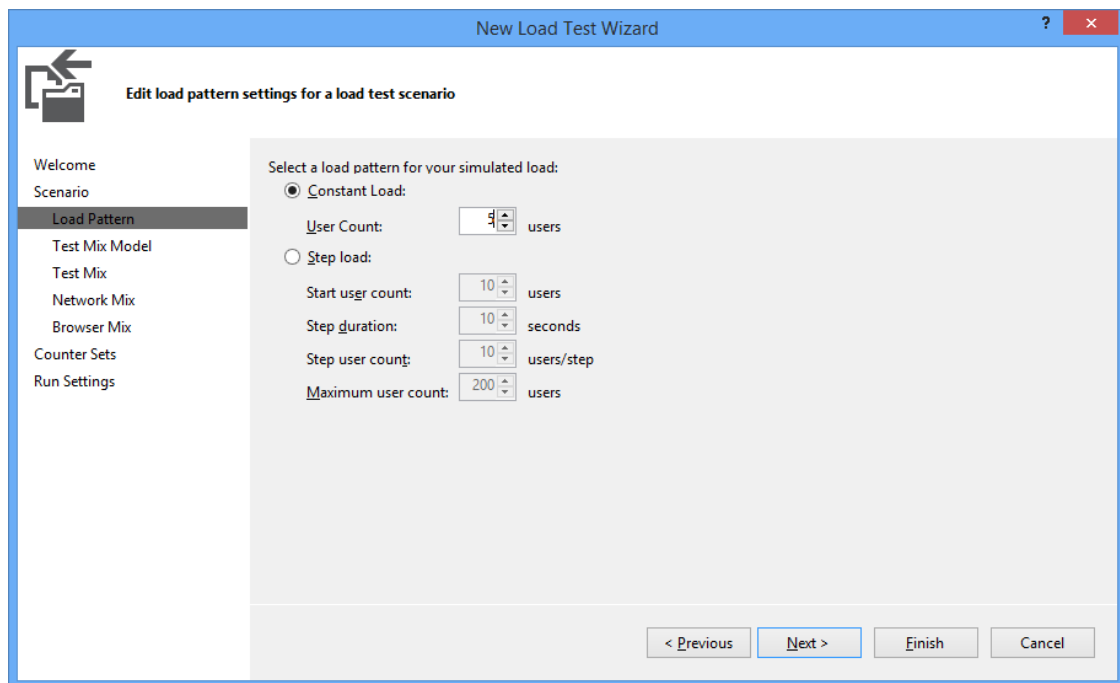
1. Right-click the solution, select **Add** and click **New Project**.
2. Select the **Web Performance and Load Test Project** and click **OK**.



3. Delete the **WebTest1** web test.
4. Right-click the project and select **Add, Load Test**



5. In the welcome screen of the wizard, click **Next**
6. In the Scenario screen, accept the default values and click **Next**
7. In the Load Pattern tab, select a Constant Load that matches the number of devices you have and click **Next**



8. In the Test Mix Model screen, accept the default values and click **Next**
9. In the **Test Mix**, click **Add** to add your unit test.
10. In all other screens, accept the default values.

Your load test is now ready. To execute the load test, click the **Run Load Test** button.

6.6. Run your tests cases from Microsoft Test Manager

To run your test cases from Microsoft Test Manager, please follow these steps:

1. Link your unit tests to test cases in Microsoft Test Manager by following the instructions from [Automate a test case in Microsoft Test Manager](#).
2. Your tests will execute on the Microsoft Test Agent. For more information on configuring Microsoft Test Agents, see [Installing and Configuring Test Agents and Test Controllers](#).
3. Each test agent will attempt to connect to a local device manager. You need to:
 1. Install M-eux Test on the same machine as the Test Agent
 2. Ensure the Device Manager is running on that machine
 3. The devices are connected to the Device Manager
 4. The Test Agent is running as Local System

6.7. Run your tests cases from Microsoft Team Build

You can run your test cases during a Team Build. To do so, please follow these steps:

1. Set up your build server. You can use an on-premise or hosted build server. For more information, see [Deploy and configure a build server](#).
2. The build agent will attempt to connect to a local device manager. You need to:
 1. Install M-eux Test on the same machine as the Build Agent
 2. Ensure the Device Manager is running on that machine
 3. The devices are connected to the Device Manager
 4. The Build Agent is running as Local System

Please note that M-eux Test does deploy your application to the device. If you want to build and deploy your app before you execute your tests, you will need to customize the build workflow within Team Build.

Chapter 7: Using M-eux Test in any .NET Project

You can call M-eux Test methods from any .Net project. This appendix explains how to call M-eux Test object and methods from any .Net project which is also called the API approach. The sample used in this appendix is written in C#. The other .Net languages can also be used.

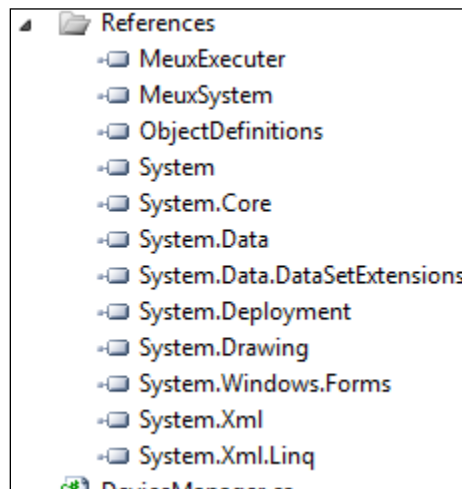
By using the API, the QA engineer can program his automation step in any .Net project. Recording is not supported in this configuration and adding objects by learn GUI or by the spy are also not supported.

7.1. Assemblies

Following assemblies needs to be included in order to use our API from your .Net project:

- MeuxExecuter.exe: This assembly contains the object to connect to the device manager and to disconnect from the device manager program.
- MeuxSystem.dll: This assembly contains the kernel to execute an automation step.
- ObjectDefinitions.dll: This assembly contains the definitions of all test objects and their methods as described in the 'Function Reference' of M-eux Test.

The three assemblies are located in the bin directory of the M-eux Test installation folder. Add the three assemblies as a reference to your Visual Studio project:



7.2. Connect the .Net executable to the device manager

The test automation steps can only be executed if a connection to the device manager is made. This is done by using the 'ScriptExecution' class of the MeuxExecuter.exe assembly.

The class 'ScriptExecution' contains two methods:

Method	Description
OpenRunEnvironment()	This method will establish the connection to the device manager. Only after calling this method, test automation methods can be called. Exceptions will be thrown if OpenRunEnvironment() is called when the connection is already made, i.e. when the method is called a second time while the connection of the first time is not yet closed.
CloseRunEnvironment()	This method will close the connection to the device manager. Exceptions will be launched if no connection was open.

Sample code:

```
ScriptExecution scriptExecution = new ScriptExecution();
scriptExecution.OpenRunEnvironment();
// Execute methods from different APIcript based classes.
scriptExecution.CloseRunEnvironment();
```

7.3. Programming automation steps

7.3.1. The APIScript class

Automation methods can only be called inside classes that are derived from the APIScript class. The API Script class is defined in the MeuxSystem assembly.

Sample code:

```
using MeuxSystem;

namespace TestDirectAPIAccess
{
    class testCase1 : APIScript
    {
        public testCase1() : base()
        {
        }

        public void execute1()
        {
        }
    }
}
```

7.3.2. Automation methods

The automation methods are called on instances of classes defined in the ObjectDefinitions assembly.

7.3.3. Object Pool

One way to access these classes is to create your own object pool. This is done by creating classes derived from the objectDefinitions classes. The main class is either the DeviceManager class to access the information in the device manager program or the MobileDevice class to connect and execute commands against the UI displayed on a connected device.

In order to access the device manager, create a class called DeviceManager which is a subclass of the ObjectDefinitions.DeviceManager class and specify descriptive attribute name.

Sample Code:

```
using MeuxSystem;

namespace TestDirectAPIAccess
{
    public sealed class DeviceManager : ObjectDefinitions.DeviceManager
    {
        internal DeviceManager(ScriptObject parent)
            : base(parent)
        {
            PPName.Value = "DeviceManager";
            AddDescriptionProperty(PPName);
        }
    }
}
```

In order to access the UI on the device, specify the device, the descriptive attributes of the window and the UI object.

Sample Code:

```
using MeuxSystem;

namespace TestDirectAPIAccess
{
    public sealed class Jamo_iphone_5 : ObjectDefinitions.MobileDevice
    {
        internal Jamo_iphone_5(ScriptObject parent)
            : base(parent)
        {
            PPName.Value = "iphone";

            AddDescriptionProperty(PPName);

            ios_UICatalog6 = new Ios_UICatalog6(this);
        }
    }
}
```

```

public readonly    Ios_UICatalog6 ios_UICatalog6;
public sealed class Ios_UICatalog6 : ObjectDefinitions.iosWindow
{
    internal Ios_UICatalog6(ScriptObject parent)
        : base(parent)
    {
        PPApplication      .Value  = "UICatalog6_1";

        AddDescriptionProperty(PPApplication      );

        ios_tableView_UICatalog = new Ios_tableView_UICatalog(this);
    }

    public readonly    Ios_tableView_UICatalog ios_tableView_UICatalog;
    public sealed class Ios_tableView_UICatalog : ObjectDefinitions.iosTableView
    {
        internal Ios_tableView_UICatalog(ScriptObject parent)
            : base(parent)
        {
            PPClassname      .Value  = "UITableViewController";
            PPNavigationBarTitle.Value  = "UICatalog";

            AddDescriptionProperty(PPClassname      );
            AddDescriptionProperty(PPNavigationBarTitle);
        }
    }
}
}
}

```

In order to use the device manager and the device, you have to create an instance and register this instance. Best practice is to do this in the constructor of you API Script based object. The instances can also be a variable inside a method and registration can also be done inside this method.

The following sample illustrates the registration. The execute1 method contains one automation step.

```

using MeuxSystem;

namespace TestDirectAPIAccess
{
    class testCase1 : APIScript
    {
        public testCase1() : base()
        {
            Register(deviceManager);
            Register(jamo_iphone_5);
        }

        public readonly DeviceManager deviceManager = new DeviceManager(null);
        public readonly Jamo_iphone_5 jamo_iphone_5 = new Jamo_iphone_5(null);

        public void execute1()
        {
            jamo_iphone_5.ios_UICatalog6.ios_tableView_UICatalog.Select("#0", "#0");
        }
    }
}

```

```
}
}
```

7.4. Descriptive programming

When applying the object pool mechanism as described in the previous section, the writing out of the automation step is easy but the creation and maintenance of the object pool is more difficult since you have to maintain it without the support you have in a normal C# based M-eux Test project. Instead of applying the object pool, you can also use descriptive programming. By applying descriptive programming, you define the objects you need in the test script and not in a separate object. You can make abstraction of the hierarchy of the UI by using the childObjects method.

The following example shows how to read in an external XML which contains the definition of a mobile web page, the id of the web element and the action to be executed against the web element.

```
public void Execute()
{
    // Create the propList to ask for the children of the web page
    // We will look for one child with a specific id attribute
    ArrayList propList = new ArrayList();
    Property idProp = new Property("id", "zipcode");
    propList.Add(idProp);
    // When looking for the child, we want to look down the full hierarchy of the DOM of
the web page
    androidphone.ConfigParamSet("childObjects.recursive", "true");
    // open the XML
    XmlDocument xd = new XmlDocument();
    xd.Load("C:\\TestFile.xml");
    XmlNodeList nodelist = xd.SelectNodes("/UATAutomation/Step"); //get all Step nodes
    foreach (XmlNode node in nodelist)
    {
        String webElementid = node.Attributes.GetNamedItem("id").Value;
        String action = node.Attributes.GetNamedItem("action").Value;
        String webPageId = node.Attributes.GetNamedItem("PageId").Value;
        String actionArg = node.InnerText;

        // set the id of the web page
        androidphone.avw_BrowserActivity.avw_avwWebView.mo_webPage.SetTOPProperty("id",
webPageId);
        // check if the page is opened. Use the default timeout for this.
        if (androidphone.avw_BrowserActivity.avw_avwWebView.mo_webPage.Exists())
        {
            // look for the web object
            idProp.Value = webElementid;
            ResolvedObject[] children =
androidphone.avw_BrowserActivity.avw_avwWebView.mo_webPage.ChildObjects(propList);
            if ((children != null) && (children.Length == 1))
            {
                try
                {
                    if ((actionArg != null) && (actionArg.Length > 0))
                    {
                        String[] args = new String[1];
                        args[0] = actionArg;
                    }
                }
            }
        }
    }
}
```

```
        children[0].Run(action, args);
    }
    else
        children[0].Run(action, new string[] { });
    }
    catch (Exception ee)
    {
    }
    }
}
}
```


Chapter 8: Summary

Now you should be able to start with the testing on Windows Phone.

We wish you a good journey with your test adventure using M-eux Test tool. If you are facing any issues with our tool, or having trouble about making your test scripts for your test cases, you can always contact us at support@jamosolutions.com.