

- Rapise, Visual Studio Test Explorer and Visual Studio Team Services
  - Integration
    - Unit Test Mapping
    - Parameters
  - Visual Studio Test Explorer
  - Visual Studio Team Services
    - Windows Agent for Test Execution
  - References

# Rapise, Visual Studio Test Explorer and Visual Studio Team Services

## Integration

### Unit Test Mapping

Rapise integrates with Visual Studio at [Unit Test](#) level.

Create a Unit Test project in Visual Studio, add a unit test and a test method. In the References section add the DLL:

```
c:\Program Files (x86)\Inflectra\Rapise\Extensions\UnitTesting\VSUnit\SeSVSUnit\Bin\Release\SeSVSUnit.dll
```

In a test method specify absolute path to a Rapise test and pass `TextContext` parameter to `Rapise.TestExecute` function:

```
namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        public TestContext TestContext { get; set; }

        [TestMethod, TestCategory("browser")]
        public void CreateNewBook()
        {
            Rapise.TestExecute(@"c:\Demo\Framework\CreateNewBook\CreateNewBook.sstest", TestContext);
        }
    }
}
```

### Parameters

To pass parameters to Rapise test create `.runsettings` file.

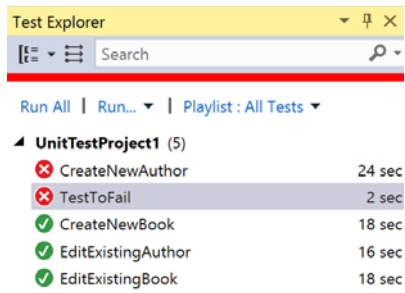
Each parameter with name starting with `g_` will be passed to Rapise via command line.

Here is an example of selecting a browser to use for execution of cross-browser tests:

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <!-- Parameters used by tests at runtime -->
  <TestRunParameters>
    <Parameter name="g_browserLibrary" value="Chrome HTML" />
  </TestRunParameters>
</RunSettings>
```

## Visual Studio Test Explorer

Once Rapise tests are mapped to unit tests one can use Visual Studio Test Explorer to run tests and analyze results.



### TestToFail

Source: [UnitTest1.cs line 38](#)

Test Failed - TestToFail

**Message: Assert.AreEqual failed.**

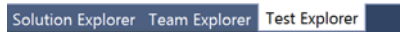
**Expected:<0>. Actual:<1>. Test passed: c:\Demo\Framework\NewTest\NewTest.sstest**

Elapsed time: 2 sec

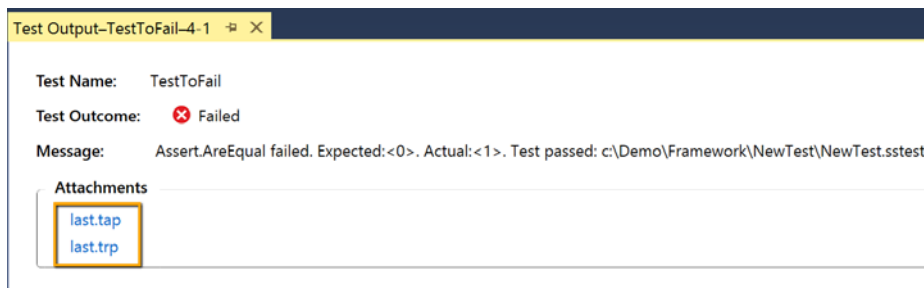
[Output](#)

Stack Trace:

Rapise.TestExecute(String path, TestContext  
UnitTest1.TestToFail())

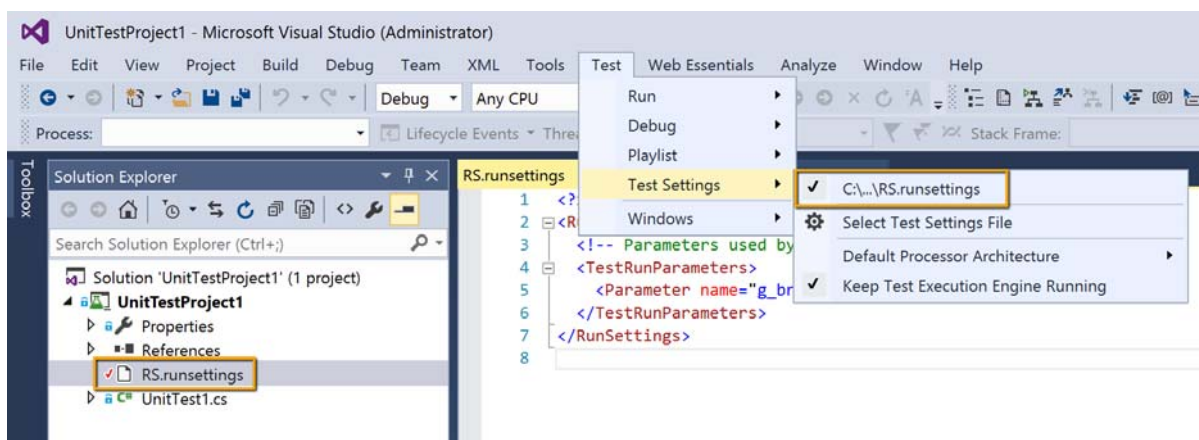


Press Output link (highlighted) to view test run results.



- last.tap - is a test report in [Test Anything Protocol](#) format (human readable). Click to open in any Text Viewer/Editor.
- last.trp - is a test report in Rapise format. Click to open in Rapise.

One can apply .runsettings file to use for execution:



## Visual Studio Team Services

In Visual Studio Team Services one can [run unit tests after making a build](#).





MyFirstProject Home Code Work Build & Release Test

Builds Releases Library Task Groups Explorer







## Build Definitions

Mine All Definitions Queued XAML



Requested by me

	<b>Build and Test in Chrome</b> : #20161207.2 requested 2 weeks ago	...	 succeeded
	<b>Build and Test in IE</b> : #20161206.9 requested 2 weeks ago	...	 succeeded







Build definition contains predefined steps:

-  NuGet restore \*\*\\*.sln  
*NuGet Installer*
-  Build solution \*\*\\*.sln  
*Visual Studio Build*
-  Test Assemblies \*\*\\$(BuildConfiguration)\\*test\*.dll;-\*\obj\\*\*  
*Visual Studio Test*
-  Publish symbols path:  
*Index Sources & Publish Symbols*
-  Copy Files to: \$(build.artifactstagingdirectory)  
*Copy Files*
-  Publish Artifact: drop  
*Publish Build Artifacts*

Here is an example configuration of the Test Assemblies step:

**Test Assemblies** \*\*\\$(BuildConfiguration)\\*test\*.dll;-\*\obj\\*\*  Version 1.\* 

**Execution Options**

Test Assembly	**\\$(BuildConfiguration)\*test*.dll;-*\obj\**	
Test Filter criteria	TestCategory=browser	
Run Settings File	UnitTestProject1\UnitTestProject1\RS.runsettings	... 
Override TestRun Parameters	g_browserLibrary=Chrome HTML	
Code Coverage Enabled	<input type="checkbox"/>	
Run In Parallel	<input type="checkbox"/>	

Advanced Execution Options

Reporting Options

- Test Assembly field contains a wildcard mask that selects unit tests from matching DLLs only
- In Test Filter criteria one can select tests by `TestCategory` which is an attribute of a `Test Method` :

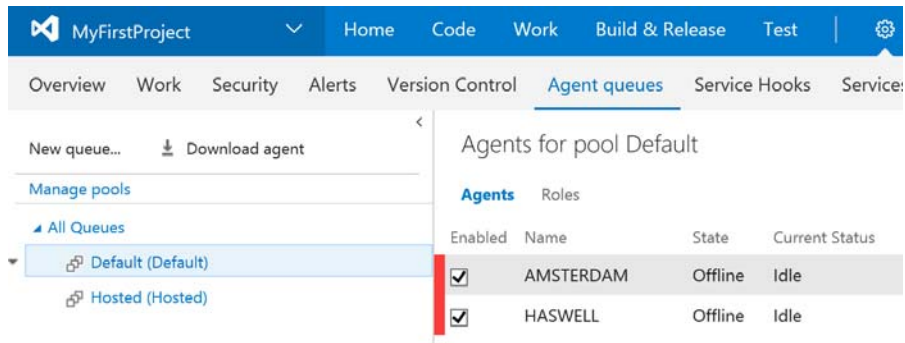
```
[TestMethod, TestCategory("browser")]
public void CreateNewBook()
{
    Rapise.TestExecute(@"c:\Demo\Framework\CreateNewBook\CreateNewBook.sstest", TestContext);
}
```

- `Run Settings File` is a link to `.runsettings` file.
- In `Override TestRun Parameters` one can override values of the parameters in `.runsettings` file.

## Windows Agent for Test Execution

VSTS can run tests in a hosted environment, but it does not contain Rapise. So most likely you will need to run tests inside your computer network. Download and connect [Windows Agent](#).

One can configure several agent pools to run tests in different environments:



The screenshot shows the VSTS interface for managing agent queues. The top navigation bar includes 'MyFirstProject', 'Home', 'Code', 'Work', 'Build & Release', and 'Test'. The main navigation bar includes 'Overview', 'Work', 'Security', 'Alerts', 'Version Control', 'Agent queues', 'Service Hooks', and 'Service:'. The 'Agent queues' page is active, showing a list of agent pools for the 'Default' pool. The list includes 'Default (Default)' and 'Hosted (Hosted)'. The 'Default (Default)' pool is selected, and its details are shown in a table below.

Agents for pool Default			
Agents		Roles	
Enabled	Name	State	Current Status
<input checked="" type="checkbox"/>	AMSTERDAM	Offline	Idle
<input checked="" type="checkbox"/>	HASWELL	Offline	Idle

## References

1. Rapise
2. Visual Studio Test Explorer
3. Visual Studio Team Services