|  |
| --- |
| Python Tools for Visual Studio |
| End-to-End Scenario Tests |
| A detailed walkthrough guide |

|  |
| --- |
| John S Costello  v1.1, updated 1/6/2012 |

# Test Environment

### Windows versions supported

Because of limited test resources, the Microsoft test team is only testing on Windows 7 x64. The product itself supports any OS which Visual Studio 2010 installs on. We are interested in expanding the test matrix to include Windows XP x86, and especially appreciate community test effort that would help us validate PTVS on XP and other OS versions.

## Visual Studio and Python configurations

The *Test Strategy Guide* in this directory has a detailed discussion of the PTVS configurations test matrix, but the key table is this one:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Basic x86 | Basic 2.7 Int. Shell | Basic 3.2 Int. Shell | Basic IPy Int. Shell | Basic x64 | Kitchen Sink |
| VS2010 Ultimate SP1 | ■ |  |  |  | ■ | ■ |
| VS2010 Integrated Shell |  | ■ | ■ | ■ |  |  |
| Python 2.5 | ■ |  |  |  |  | ■ |
| Python 2.5 x64 |  |  |  |  | ■ |  |
| Python 2.6 | ■ |  |  |  |  | ■ |
| Python 2.6 x64 |  |  |  |  | ■ |  |
| Enthought Python 2.7 | □ | □ | ■ | ■ |  | ■ |
| Enthought Python 2.7 x64 |  |  |  |  | □ |  |
| Python 3.0 | ■ |  |  |  |  | ■ |
| Python 3.0 x64 |  |  |  |  | ■ |  |
| Python 3.1 | ■ |  |  |  |  | ■ |
| Python 3.1 x64 |  |  |  |  | ■ |  |
| ActiveState Python 3.2 | □ | ■ | □ | ■ |  | ■ |
| ActiveState Python 3.2 x64 |  |  |  |  | □ |  |
| IronPython 2.7 | □ | ■ | ■ | □ | □ | ■ |
| PyPy | ■ |  |  |  |  | ■ |
| Visual Studio plugins? |  |  |  |  |  | ■ |
| Lots of different SDKs? |  |  |  |  |  | ■ |
| Other things that can cause interop problems …? |  |  |  |  |  | ■ |

The key configuration is highlighted – all the scenarios should be tested in this configuration. Some subset of scenarios should be tested in the other configurations as well; ideally, we would run through each scenario two times: once on the “Basic x86” and once in one of the other four configurations. The “Kitchen Sink” configuration is for future planning and interop scenarios.

The “Basic x86” configuration is the one recommended for day-to-day testing; this is how test and dev should have their workstations configured. The other configurations can be tested inside a virtual machine.

# Scenarios

## Installation (v1.0, v1.1)

### Initial state

A machine set up according to one of the columns in the [test matrix](#_Visual_Studio_and). In the most commonly tested configuration, this will be developer workstation with the following software:

* Win7 x64 Enterprise
* Visual Studio 2010 Ultimate SP1
* HPC SDK v3sp3
* Excel 2010 installed
* MSR Kinect SDK latest version installed
* CPython, x86 versions 2.5, 2.6, 3.0, 3.1 from <http://python.org>
* Enthought Python Distribution with Python 2.7 x86
* ActiveState Python Distribution with Python 3.2 x86
* IronPython 2.7 latest version from <http://ironpython.net/>
* PyPy, latest version from <http://pypy.org>

### Test procedure

Install PTVS, the PyKinect libraries, and the Pyvot libraries from the MSIs, using the default options.

### Verify

* “Python Tools” is present in the Tools menu and in Tools -> Options
* The installed CPython and IronPython interpreters are detected (PyPy will not be)
  + Their interactive windows are available in Tools -> Python Tools
  + They are available in Tools -> Options -> Python Tools -> Interpreter Options
* The proper templates are installed and available in File -> New Project
* File -> New Project -> Python Application creates a new project as expected
  + F5 debugging works on this project
* Alt-I brings up the default interactive window

>>> import sys

>>> sys.version

'2.7.2 |EPD 7.1-2 (32-bit)| (default, Jul  3 2011, 15:13:59) [MSC v.1500 32 bit (Intel)]'

## Custom interpreters (v1.0)

### Initial state

Make sure you have PyPy installed in your test environment, then complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. Open Tools -> Options -> Python Tools -> Interpreter Options
2. Add interpreter named PyPy
3. Configure the interpreter according to your PyPy installation.
4. Click Generate Completion Database
5. Create a new Python project

### Verify

* In Project -> Properties, the PyPy interpreter is available
  + Select the PyPy interpreter and save the project.
  + F5 debugging should work.
* In Tools -> Python Tools, the PyPy interactive window is available and works correctly.

>>> import sys

>>> sys.version

'Your PyPy version here'

## Solution Management (v1.1)

The next scenarios build on one another (Solution Management, Editor, Refactoring, Debugging)

### Initial state

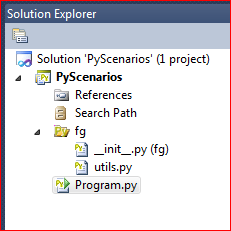
Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario. Have the Enthought Python Distribution as your default interpreter.

### Test procedure

1. Unzip the PyScenarios.zip from this directory into c:\temp\PyScenarios
2. In Visual Studio, File -> New Project from Existing Code
3. Select Python Project, Next
4. Files are in C:\temp\PyScenarios; make sure “include subfolders” is checked.
5. Name is PyScenarios, output type Console Application, Finish.
6. Alt-I to open the default interpreter window (or reset the window if it’s already open)

### Verify

* You get a project with the following structure:



## Refactoring (v1.0, v1.1): Rename

### Initial state

Complete the [Solution Management](#_Solution_Management_(v1.1)) scenario.

### Test procedure

1. In Program.py, select the “exit\_flag” variable (anywhere)
2. Refactor -> Rename, and rename to “exit\_now”

### Verify

* In the preview pane, ensure that the variable is renamed in “utils.py” as well as Program.py
* Apply the changes, then ensure that they got propagated.
* Ctrl-z and ensure that all changes are undone

## Refactoring (v1.0, v1.1): Extract Method

### Initial state

Complete the [Solution Management](#_Solution_Management_(v1.1)) scenario.

### Test procedure

1. In utils.py, select the following code:

(i + 1) % 100000000

1. Refactor -> Extract Method, and extract to <module> scope.

### Verify

* The following method is created at module scope:

def new\_method(i):

    return (i + 1) % 100000000

* Line 10 is now

       i = new\_method(i)

* Ctrl-z to undo changes, and the changes are all undone.

## Debugging (v1.0, v1.1): start debugging

### Initial state

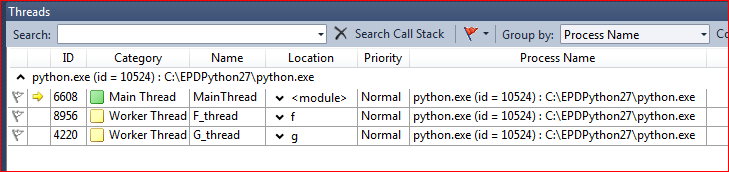
Complete the [Solution Management](#_Solution_Management_(v1.1)) scenario.

### Test procedure

1. Set Program.py as the startup file.
2. F5 to start debugging.
3. Wait about 10 seconds, then Debug -> Break All

### Verify

* Both f and g indicate that they are making progress in the console window
* In the Debug -> Windows -> Threads window, the threads have appropriate names:



* Ensure that you can look at variables in all the execution contexts of all the threads by switching threads in the Threads window and viewing the variables in the Debug -> Windows -> Locals window.

## MPI (v1.0)

See “**Writing MPI programs in Python on a Windows HPC cluster**” in the Documentation walkthrough below (<http://pytools.codeplex.com/wikipage?title=Detailed%20Walk-thu%20Guide%20-%20HPC%20and%20Cloud%20Features&referringTitle=Home>)

## IPython (v1.0, v1.1)

See “**IPython Quick Start Guide**” in the Documentation walkthrough below (<http://pytools.codeplex.com/wikipage?title=Detailed%20Walk-thu%20Guide%20-%20HPC%20and%20Cloud%20Features&referringTitle=Home>)

## Documentation (v1.0, v1.1)

Such a short scenario! But it is definitely the most time-consuming.

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario. Have the Enthought Python Distribution as your default interpreter.

### Test procedure

Walk through the documentation on the pytools codeplex site, starting on <http://pytools.codeplex.com/documentation>

### Verify

The documentation contains no errors; the product works the way the documentation says it does.

## Profiling (v1.0)

This is essentially a UI correctness verification; there are unit tests that also exercise Profiling.

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. Analyze -> Python Profiling
2. Select Standalone Script
   1. Python interpreter: your Python27 interpreter
   2. Script C:\<your python27 installation>\Lib\test\pystone.py
   3. Working directory c:\temp

### Verify

* Profiling runs to completion
* Analysis runs to completion
* The profiling data under the various report views looks sensible

## IronPython (v1.0, v1.1) – WPF application

### Initial state

Make sure you have IronPython 2.7 installed in your test environment, then complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. File -> New Project -> IronPython WPF Application
2. Set a breakpoint on line 11
3. F5 to start debugging

### Verify

* The breakpoint is hit
* When you continue, the window is shown
* When you close the window, debugging ends.

## IronPython (v1.0, v1.1) – Silverlight Webpage

### Initial state

Make sure you have IronPython 2.7 installed in your test environment, then complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. File -> New Project -> IronPython Silverlight Webpage
2. Set a breakpoint on line 2 of SilverLightPage1.py
3. F5 to start debugging
4. Click the button

### Verify

* The breakpoint is hit
* When you continue, the dialog is shown
* When you close the window, debugging ends.

## Sho (v1.0, v1.1)

This test is essentially a documentation verification test. We want to make sure that Sho users have a great experience when using PTVS as their IDE; although we rely on the Sho team to do in-depth testing as dogfooders, we will make sure that the initial experience is great by walking through their online how-to.

### Initial state

Make sure you have IronPython 2.7 installed in your test environment, then complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

Install Sho and configure it for use with PTVS by following the steps in the “Using Sho with PTVS” document on the Sho website: <http://research.microsoft.com/en-us/projects/sho/ptvs.aspx>

Try the examples on the Sho Examples webpage: <http://research.microsoft.com/en-us/projects/sho/screenshots.aspx>

### Verify

* The documentation works correctly (if there is an obvious documentation error, this is a Sho bug; make a best-effort attempt to figure out what’s wrong and alert the Sho team).
* Sho is successfully installed into the PTVS REPL.
* The Sho examples work.
* There are no errors or exceptions thrown.

## Diagnostic Info (v1.0, v1.1)

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. Select Tools -> Python Tools -> Diagnostic Info

### Verify

* The information displayed is sane.
* The performance is acceptable (no VS hangs)

## REPL (v1.0, v1.1)

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. Create a new Python Application
2. Alt-I to bring up the default REPL.

### Verify

* Import <space> gives intellisense for modules in the REPL
* Debug -> Execute Project in Python Interactive executes the project
* $help brings up help for the repl-specific commands
* Clicking on the “clear” and “reset” buttons works as expected.

## Pyvot (v1.1)

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario.

### Test procedure

1. Tools -> Samples -> Pyvot , install into your default interpreter
2. File -> New Project -> Pyvot Excel Sample

### Verify

* Execute project in interactive works as expected (the code runs, Excel is displayed, no exceptions or other failures)
* Importing xl in the REPL then allows intellisense on the xl module.

## PyKinect (v1.1)

### Initial state

Complete the steps in the [Installation](#_Installation_(v1.0,_v1.1)) scenario. Have a Kinect device connected to your machine and turned on.

### Test procedure

1. Tools -> Samples -> PyKinect , install into your default interpreter
2. Open the sample PyKinect application in PTVS
3. F5 to start debugging

### Verify

* Skeleton tracking works as expected.
* Image acquisition works as expected.

# Future Scenarios (planning)

These are all TBD ☺

## Ditertools

## Win8