Debugging DS-5 Jython Scripts

Application Note 351

Non-Confidential

Debugging DS-5 Jython Scripts

Application Note 351

Copyright © 2013 ARM. All rights reserved.

Release Information

Proprietary Notice

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

1 Introduction

This application note describes how to debug DS-5 Jython scripts using the PyDev debugger included in DS-5.

This works by using the PyDev remote debugger. This consists of a debug server, which runs inside eclipse, and the debug client, which runs inside your script's interpreter.

Note that, while these instructions assume that you are using the PyDev debugger, it should be possible to use any other Python/Jython remote debugger. However, note that most debuggers are designed for Python, and may not work perfectly with Jython.

It is also worth noting that the PyDev debugger is not limited to debugging DS-5 Jython scripts, and should also be able to debug other Python and Jython scripts. However, if a script is launched from outside DS-5, the PyDev debug client may not be in the python path by default. The PyDev debug client can be found at `eclipse/plugins/org.python.pydev.debug_<version>/pysrc`, inside your DS-5 installation directory.

The PyDev debugger client uses the `sys.settrace()` function to set a tracing function which allows it to observe and pause code execution, so scripts should not use the `sys.settrace()` function for their own tracing purposes when the debugger is connected.

2 Configuring your DS-5 development environment

The first task is to set up your development environment for using the PyDev remote debugger. This is done by firstly enabling the **PyDev Debugger** command group, which allows you to start and stop the debug server, and secondly opening the **Debug** perspective, if it is not already open.

1. Select **Window → Customize Perspective…**

2. Switch to the **Command Groups Availability** tab

3. In the **Available Command Groups** box, enable the **PyDev Debug** option and click OK
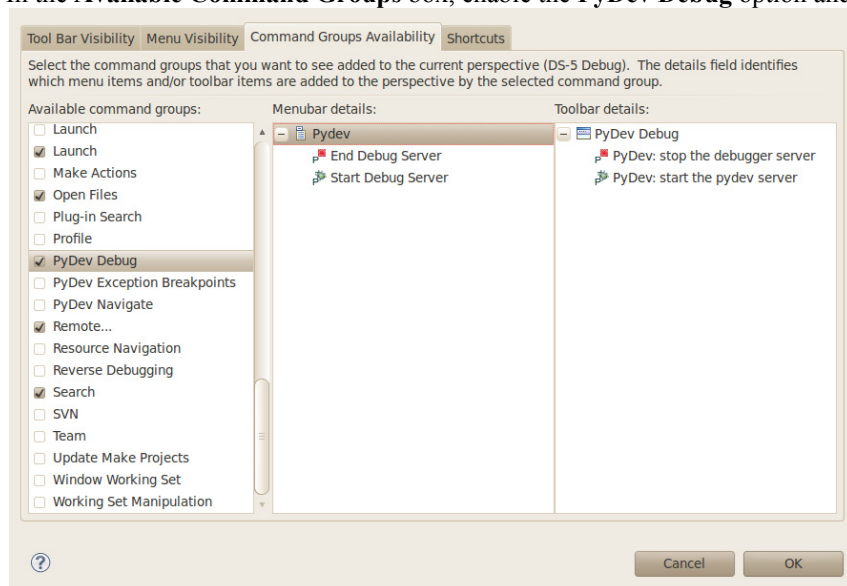
4. Select **Window → Open Perspective → Other…**

5. Select **Debug** (not **DS-5 Debug**), and click OK

Once the Debug perspective has been opened, you can return to your previous perspective.

3 Starting the debug server

Next you need to start the PyDev debug server. This should remain open until you close DS-5, even when you disconnect from all targets. The PyDev debug server shows up in the **Console** view, which shows its current state and allows it to be stopped if needed.

1. Select **PyDev → Start Debug Server**

2. The debug server should start in the **Console** view
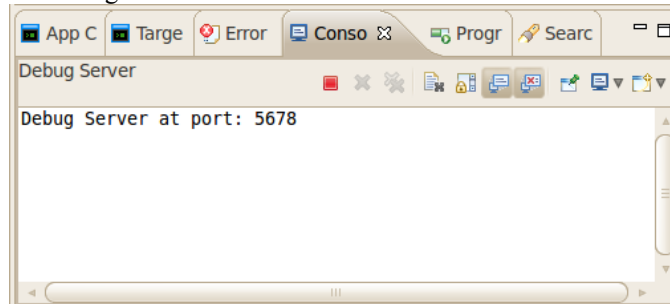
4 Modifying the script

You now need to modify the python script to be debugged, importing the debug client and connecting it to the server.

1. Open your Jython script, and insert the following lines near the top of the file:
   ```
   import pydevd
   pydevd.settrace()
   ```

2. Optionally, the second line can be modified, to cause all output from the Jython script to be copied to the debug server console in the **Console** view:
   ```
   pydevd.settrace(stdoutToServer=True, stderrToServer=True)
   ```

When the second line is executed, the debug client connects to the debug server and pauses execution of the Jython script.

# 5  Debugging the script

The script can now be started as normal, and when `pydevd.settrace()` is executed, the Python interpreter searches for Python debugger to connect to. Since you started a Python debugger earlier, execution of the script is halted and transferred to the Python debugger. Note that because execution is halted, any progress monitor dialog for the script should be backgrounded to ease switching to the Python debugger and interacting with it.
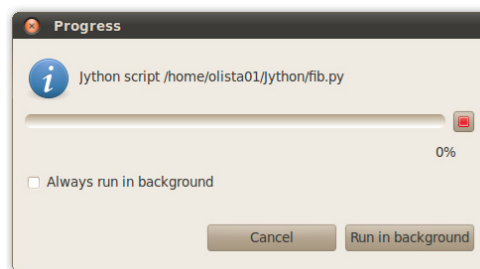


Also, when debugging a DS-5 Jython script, the following error lines appear in the DS-5 console; those can be safely ignored.

```
ERROR(LOG123): --
ERROR(LOG123): pydev debugger: CRITICAL WARNING: This version of python seems to be incorrectly compiled (internal generated filenames are not absolute)
ERROR(LOG123): pydev debugger: The debugger may still function, but it will work slower and may miss breakpoints.
ERROR(LOG123): pydev debugger: Related bug: http://bugs.python.org/issue1666807
ERROR(LOG123): --
ERROR(LOG123): pydev debugger: Unable to find real location for: __pyclasspath__/Lib/threading.py
8
ERROR(LOG123): pydev debugger: Unable to find real location for: __pyclasspath__/Lib/atexit.py
ERROR(CMD656): The script /home/olista01/Jython/fib.py failed to complete due to an error during execution of the script
```

It is now possible to switch to the **Debug** perspective to debug the Jython script. In this perspective, the standard eclipse debugging views can be used to debug the Jython code.

The **Debug** view has controls to resume, pause and step the code, as well as the ability to switch between threads and view the stack of each thread.



The **Variables** view shows the values of all local and global variables, and allows them to be modified when the program is suspended. Note that modifying variables will currently only work in the outermost stack frame, due to a limitation in Python and Jython. Variables that were modified the last time the program stepped are shown highlighted in yellow.
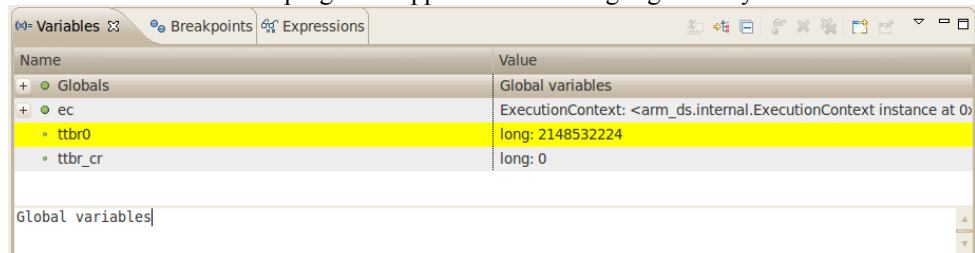


Breakpoints can be set by double-clicking in the left margin, and all current breakpoints are shown in the **Breakpoints** view. From the breakpoint view, you can remove or temporarily disable any breakpoint in your code. By right-clicking on a breakpoint in the left-hand margin (not in the **Breakpoints** view) and selecting **Breakpoint Properties...**, you can set a condition, so that the breakpoint is ignored unless the condition evaluates to true.

# 6   Debugging DTSL scripts

The PyDev debugger can also be used to debug DTSL scripts, but some additional considerations must be taken into account. When a DTSL script is executed, it runs to completion, but only creates classes which are later instantiated by DS-5. Because of this, the PyDev debugger should not be connected at the top of the file, but instead to the beginning of the `__init__` method. This means that the debugger will be connected when the DTSL object is created. Other than that, DTSL scripts can be debugged as usual.

```python
class OMAP35xx(DTSLv1):
    def __init__(self, root):
        import pydevd
        pydevd.settrace()

        DTSLv1.__init__(self, root)

        dapDev = self.findDevice("ARMCS-DP")
        self.dap = CSDAP(self, dapDev, "DAP")
```

7   Debugging flash load scripts

Debugging flash load scripts is similar to debugging DTSL scripts, in that the script is executed once to load a class, and this class is then later instantiated and its methods are called. Again, `pydevd.settrace()` must be called inside the methods of the class which inherits from `FlashMethodv1`, not at the top of the script. Typically these are `getDefaultRegions()`, `getDefaultParameters()`, `setup()`, `teardown()` and `program()`, as these are the methods that can be called by DS-5, though it is possible to connect the debugger at any point in the script.