TestManager – Python API

RedRat Ltd

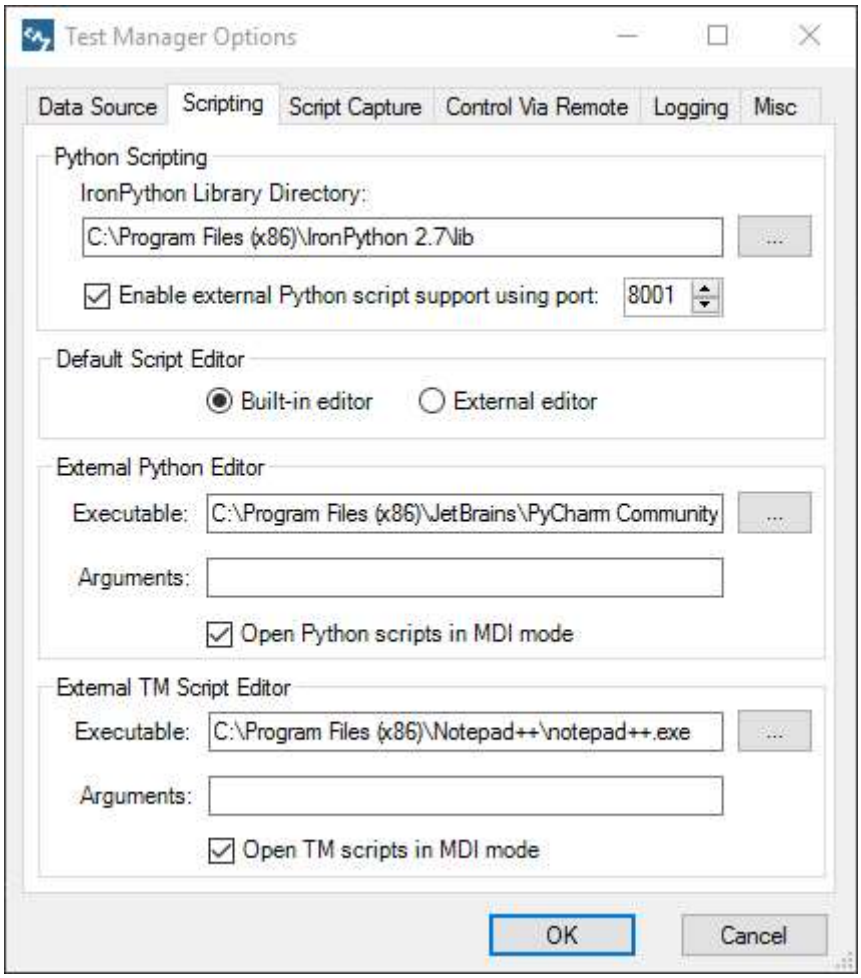
December 2015

For TestManager Version 4.61

Contents

1. Introduction

TestManager supports scripting using Python, as implemented in IronPython 2.7. Much of the Python standard library is not included with TestManager and must be installed separately, if required. Once IronPython is installed on your machine you must update the options within TestManager so that the IronPython library directory can be located, as illustrated in the following screenshot:



IronPython scripts can be run from either inside TestManager, or from outside (TestManager V4.61 onwards).

2. TestManager Python environment

The Python environment within TestManager is much the same as that provided by IronPython, but with two important differences. Firstly, TestManager defines a global object *tm* to allow access to the TestManager API. Secondly, the import mechanism has been extended to allow scripts defined within TestManager to be imported as normal Python modules.

3. Importing scripts from TestManager

All TestManager scripts are encapsulated within the pseudo-package *scripts* in the same hierarchy as displayed by TestManager. Consider the example where we have a script called utils within the Python folder containing the following code:

```
def foo():
      print 'Hello, World!'
```

We can call *foo* like so:

```
import scripts.Python.utils as utils
utils.foo()
```

Notice that the names of the folders and scripts involved must be valid Python module names for the import to succeed, but TestManager does not enforce this. Note also that folder and script names are case sensitive.

4. Using IronPython Outside TestManager

Python scripts can also be executed outside TestManager (from V4.61 onwards) in an almost identical fashion to scripts running within TestManager. This allows TestManager itself to be used as the STB and IR signal dataset management system, while gaining the benefits of working with Python scripts placed on the file system:

- Script source code management systems can be used (Git, SVN etc.)
- Python code can be written, debugged and run from within a development environment, such as PyCharm.

Python scripts running outside TestManager use .NET's Windows Communication Foundation (WCF) to communicate with TestManager, so IronPython has to be used.

4.1 Enabling External Script Execution Support

This is done in the Scripting tab of the Options dialog, enabling support and setting the port to use. The default is port 8001.

Once this is done, the computer's firewall may prompt you to allow communication on this port. It is recommended that communication is only allowed on domain or private networks.

4.2 Setting up a Python Script to use TestManager

A Python script executed within TestManager is almost identical to one operating outside TestManager, however the external execution environment has to be setup to include the necessary files. These are:

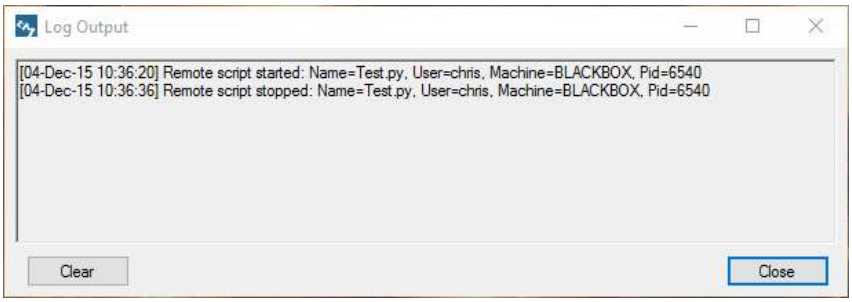
- ***TestManager.Core.dll*** – The core .NET code for the Python client to communicate with TestManager.
- ***tm.py*** – The Python module which provides the *tm* API, i.e. wraps the communication code.
- ***tmtasks.py*** – A Python class to support concurrent control of multiple STBs. See section 7.

The simplest method to use these files is to place them in the same directory as your Python script.

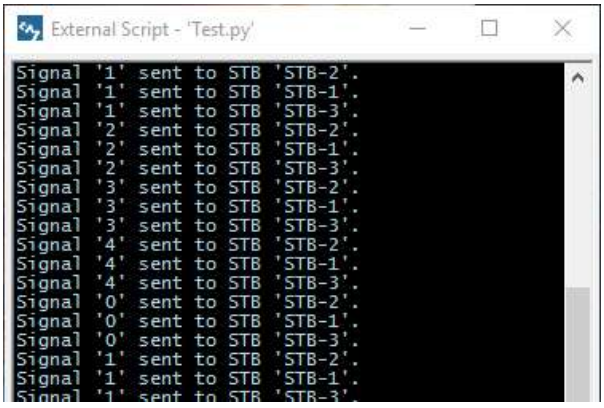
The Python client code *tm.py* needs to know how to connect to TestManager. It is setup to use the default values of machine ***localhost*** and port ***8001***. If your setup is different, then please edit these values near the top of the ***tm.py*** module.

4.3 Running an External Python Script

When an external Python script is started or stopped, this is listed in the main Log Output window:



TestManager also opens a script execution log window, in a similar fashion to internal script execution. This windows is closed as soon as the external script is terminated. Multiple external scripts will cause multiple monitor windows to open.



4.4 A Simple External Python Script

```
import tm

stbs = tm.reserve_stbs(['STB-1', 'STB-2', 'STB-3'])
tm.send('play', stbs)
tm.send('pause', stb[0])
```

The folder *ExamplePythonClient* in the TestManager installation directory shows an example Python script which can be executed with IronPython.

5. The TestManager Python API

All functions defined by TestManager are accessible through the *tm* object. The following sections document these functions and give hints on when to use them.

5.1 Data retrieval functions

We provide several functions to retrieve information about the objects defined in TestManager. These functions are not required to write simple scripts, where the name of the STB, signal etc. are already known, but can be useful when writing more complicated scripts that operate with subsets of these. For example, you might use the get_stb_names function when writing a script that uses some or all of the STBs defined in TestManager.

5.1.1 get_stb_names()

Returns the names of the STBs defined in TestManager as a list of strings.

5.1.2 get_zone_names()

Returns the names of the zones defined in TestManager as a list of strings.

5.1.3 get_signal_names()

Returns the names of the signals defined in TestManager as a list of strings.

5.1.4 get_macro_names()

Returns the names of the macros defined in TestManager as a list of strings.

5.1.5 get_named_operation_names()

Returns the names of the named operations defined in TestManager as a list of strings.

5.2 Signal output functions

Signals can be sent either to all STBs or to a subset of them. In the latter case, this subset must first be reserved, either by calling reserve_stbs or reserve_zones. Notice that both of these functions return a list of STB objects. There is no function to release STBs or zones – TestManager does this for you.

5.2.1 reserve_stbs(stb_names)

Reserves one or more STBs specified by *stb_names*, which is a list of strings, and returns a list of STB objects.

5.2.2 reserve_zone(zone_name)

Reserves the zone specified by *zone_name*, which is a string, and returns the list of STB objects in that zone.

5.2.3 send(name)

Transmits the named operation, macro or signal specified by *name*, which is a string, to all STBs.

5.2.4 send(name, duration)

Transmits the signal specified by *name*, which is a string, to all STBs for the given duration. The duration is a floating point number, specified in seconds, and should be larger than 0.1s.

The duration of an IR signal is extended by TestManager through increasing the number of signal repeats, so simulating a longer remote control button press. If the IR signal does not have repeat data, or the duration given causes the number of repeats to exceed 255, then an error will be generated. This only applies to signals, and not to named operations or macros.

5.2.5 send(name, stb)

Transmits the named operation, macro or signal specified by *name*, which is a string, to one *stb* object.

5.2.6 send(name, stb, duration)

Transmits the signal specified by *name*, which is a string, to one *stb* object for the given duration. See section 5.2.4 for more details on the duration parameter.

5.2.7 send(name, stbs)

Transmits the named operation, macro or signal specified by *name*, which is a string, to the list of STB objects *stbs*.

5.2.8 send(name, stbs, duration)

Transmits the signal specified by *name*, which is a string, to the list of STB objects *stbs* for the given duration. See section 5.2.4 for more details on the duration parameter.

5.3 STB object

The STB object has the following properties, which must not be changed.

5.3.1 id

The ID of this STB, used internally by TestManager.

5.3.2 name

The name of the STB.

6. Example scripts

6.1 Send a signal to a known STB

```
stbs = tm.reserve_stbs(['My STB'])
tm.send('play', stbs)
```

6.2 Send a signal to all STBs except one

```
all_stbs_names = tm.get_stb_names()
my_stbs = [stb for stb in all_stbs_names if stb != 'STB-8']
stbs = tm.reserve_stbs(my_stbs)
tm.send('play', stbs)
```

6.3 Send a signal to a zone

```
rack1 = tm.reserve_zone('Rack 1')
tm.send('play', rack1)
```

6.4 Send every signal to a zone

```
rack1 = tm.reserve_zone('Rack 1')
for signal in tm.get_signal_names():
      tm.send(signal, rack1)
```

7. Parallel Execution of STB Control Operations

Generally, multiple STBs are controlled by sending the same sequence of control signals to a list of them. However, it can sometimes be necessary to send different sets of commands to different STBs, in effect doing multiple, different operations at the same time.

The .NET framework has a lot of support for these kind of concurrent operations, one form of this support being the *Task Parallel Library*. This can get quite complex to use, so parts of it have been wrapped my the *tmtasks.py* module to give simple but effective use in Python.

To use it in code, take the following steps:

1. Import the *tmtasks* module:
   ```
   import tmtasks
   ```

2. Create a *tmtasks* object:
   ```
   tmtasks = tmtasks.TmTasks()
   ```

3. Create a function which you want to call multiple times, for example once for each STB in a list.

4. Call this function multiple times:

   ```
   for i in range(len(stbs)):
         tmtasks.startTask( lambda x=stbs[i]: sendSeq(x))
   ```

   These functions will then be started up in the background. The slightly strange syntax: 'x=param' is needed to capture the actual parameter instance. See lambda function closures.

5. Wait for completion of all the tasks:
   ```
   tmtasks.waitAll()
   ```

When printing out information from functions being run in parallel, the text from different functions can become mixed up, i.e. is not *thread safe*. To get round this, use the tmtasks print mechanism which ensures that each line is printed sequentially:

```
tmtasks.printMessage("Starting output to " + stb.name)
```