

# Spirent TestCenter Application Note: White Box Testing

*Dr. Strange-Results, or: How I Learned To Stop Worrying and Love White Box Testing*

Chris McCoy, Product Marketing Engineer, Spirent Communications

## 1. INTRODUCTION

Spirent TestCenter release 1.20 introduces new features to enable the foundation for White Box Testing. These features are designed to allow you, the test engineer, to complete your testing quicker, with more accuracy, and more consistency than previously possible with other testing tools. While white box testing won't hold off nuclear Armageddon, it will make your testing sessions much more productive.

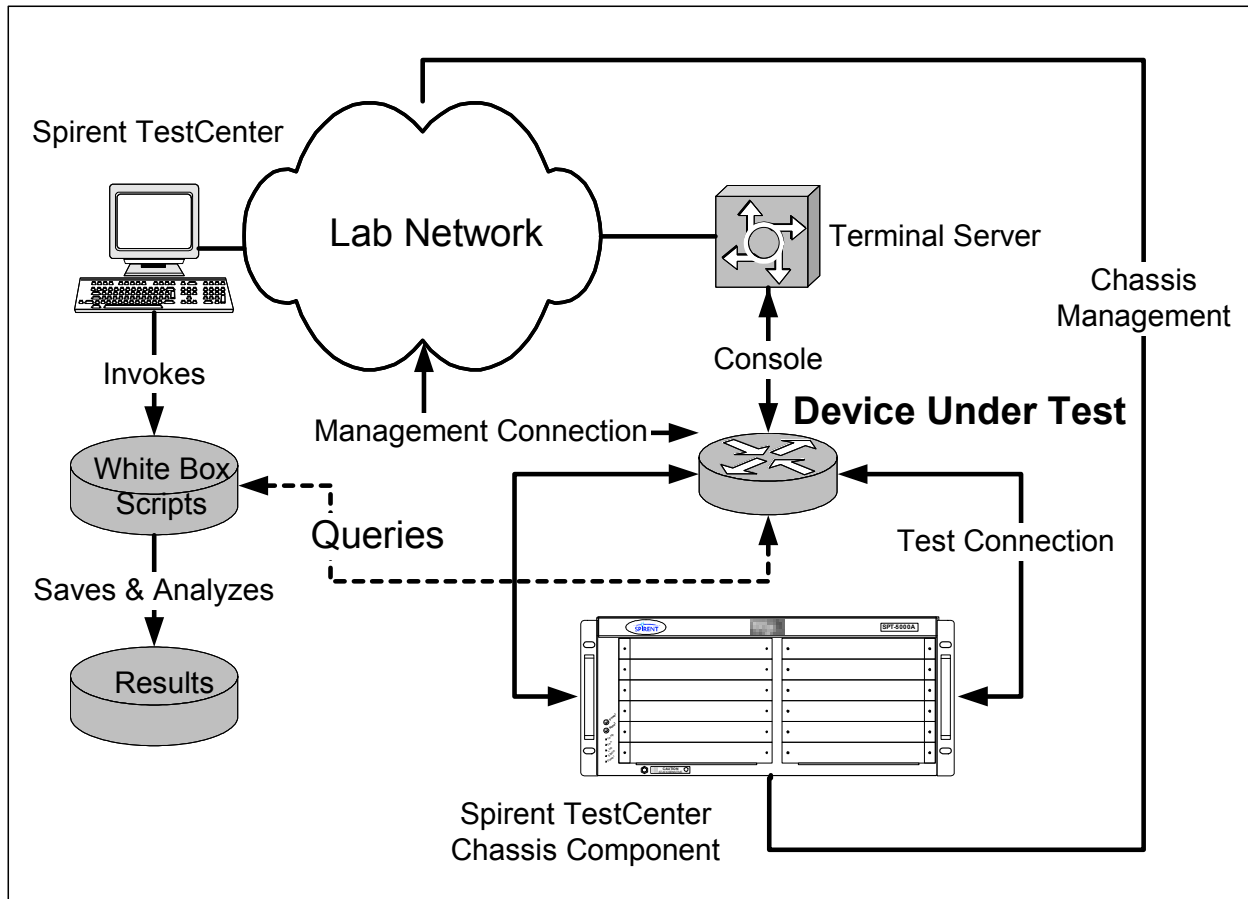
## 2. WHAT IS WHITE BOX TESTING?

White Box testing (sometimes called “clear box” or “glass box” testing) couples your test tools with your device under test (DUT). The test tool, through white box scripts, has enough intelligence to log into the DUT, retrieve results, and perform analysis. Previous test methodologies utilized “Opaque Box” or “Black Box” testing where your test tool executed completely independent from your device or system under test (SUT), leaving only you, the test engineer, to tie it all together and analyze the output. The test tool could not see into the DUT, nor could the DUT internal and external results be compared in a systematic fashion to determine if the desired behavior was really achieved. Spirent TestCenter contains new functionality to automatically perform this analysis at key locations throughout your test cycle, from within an easy-to-use GUI. White box can also assist with configuration of the DUT while the test is in progress. Analysis & configuration may be performed:

- At the beginning of the test – Configure DUT baseline, enable protocols
- At the beginning of the iteration, before sending test traffic – Reset counters
- Throughout the test iteration at determined “checkpoints” – Verify adjacencies, routes. Apply subsequent configurations.
- At the end of iteration – Look for interface errors, memory leaks, buffer overruns.
- At the end of test – Clean up

The key to white box is an out-of-band channel to the DUT. This provides the path of communication between the white box scripts and the DUT through such protocols as

Telnet, Secure Shell (SSH), SNMP, NetConf, TL1 or some other proprietary Command Line Interface (CLI). This exchange may be directly through an out-of-band Ethernet connection, or indirectly through a terminal server attached to the console of the DUT. Figure 1 shows the layout of a test bed supporting White Box testing.



**Figure 1 – The White Box Testbed**

Once this layout is in place, the possibilities are endless.

### 3. SPIRENT TESTCENTER ENABLES WHITE BOX TESTING

Spirent TestCenter, through its flap scheduler and script.ini initialization file, provides the “glue” for white box. The flap scheduler contains enhancements, shown below, that allow for execution of external processes from within the GUI. Other enhancements also include:

- Establishment of routing sessions/adjacencies without advertising routes allowing for step-wise verification of sessions before blasting the DUT with multiple routes
- Later advertising those routes in a new step
- User-specified delays between events in a single step

- Repeating a step multiple times
- Adding comments & names which will show up in the chart graphics when an event executes
- Making a step “blocking” or “nonblocking,” explained later in this document

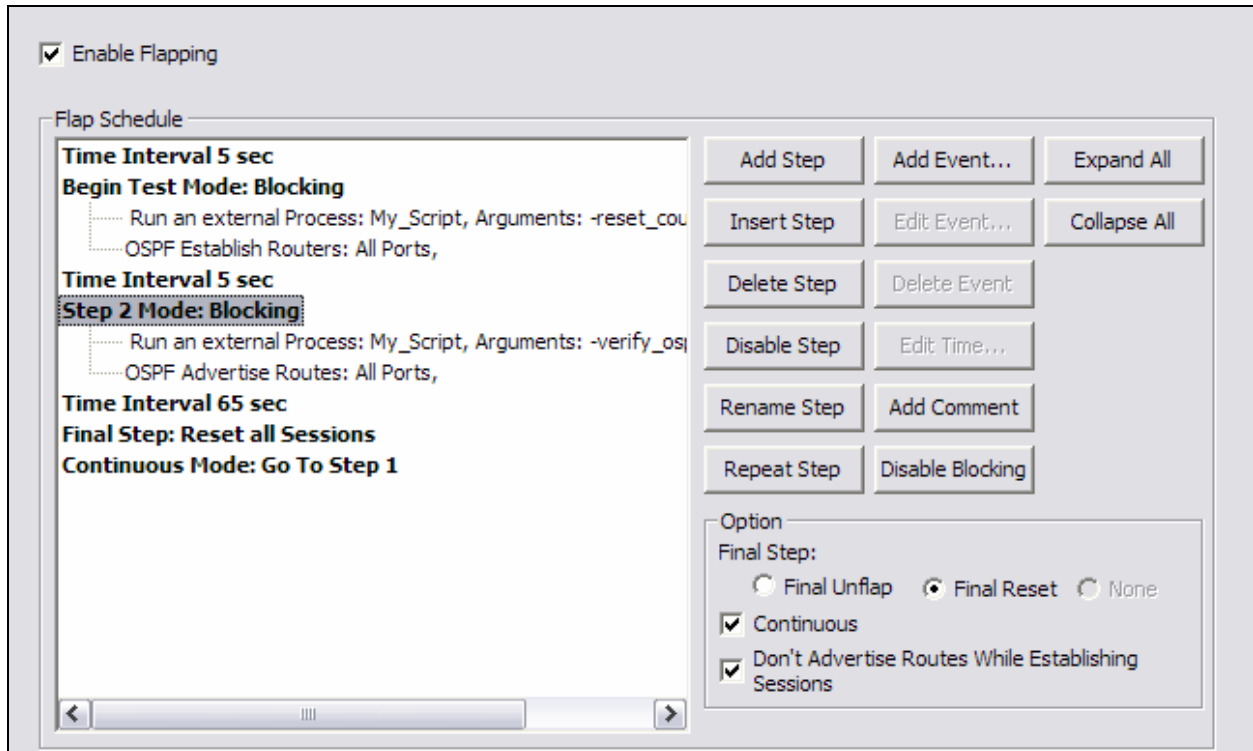


Figure 2 – The Enhanced Flap Scheduler

### 3.1 White Box Scripts

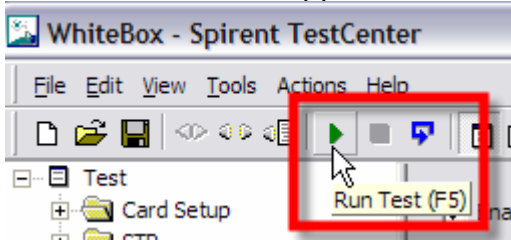
White box scripts are any program, executable or scripts that can be executed from the Windows operating system. Spirent TestCenter is script-agnostic, and allows the proper tool for the job which it is most suitable. These scripts can be any of those shown below (not at all inclusive):

- Batch (.bat) files
- Tcl/Expect scripts
- Tk GUI scripts
- SNMP gets/sets
- Python scripts

To invoke these scripts from the flap scheduler, we must first add them to the script.ini file, a text file which is contained in the Spirent TestCenter install directory (usually C:\Program Files\Spirent Communications\Spirent TestCenter\Spirent TestCenter Application). A sample script.ini file has already been created for you in this directory.



**Tip:** The script.ini file is read each time the Start Test button is pressed. Your white box framework may even modify this file on the fly before the test to change the behavior of the application without changing your configuration



Below is a sample script.ini file.

```
# sample ini file
[scripts]
#script identifier, blocking or not, timeout(ms) if blocking
view_routes 1 500
#command line to be executed
tc\sh84.exe view_routes.tc\

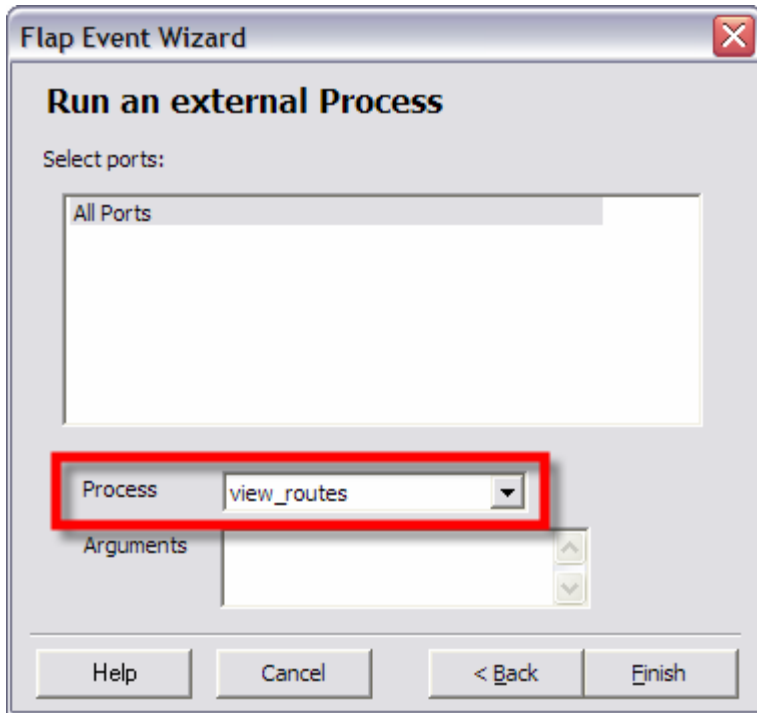
[attributes]
RunScriptBeforeStartOfTest 0
RunScriptBeforeStartTraffic 0
RunScriptAfterEndOfTest 0
RunScriptAfterEachTestIteration 0
StartOfTestScript startoftest
BeforeStartTrafficScript starttraffic
EndOfTestScript endoftest
AfterEachTestIterationScript eachiteration
```

**Figure 3 – Script.ini format**

The script.ini file is divided up into two sections, the [scripts] section and the [attributes] section. Any line that begins with a hash symbol '#' is considered a comment and ignored by the application. First, let's talk about the scripts section.

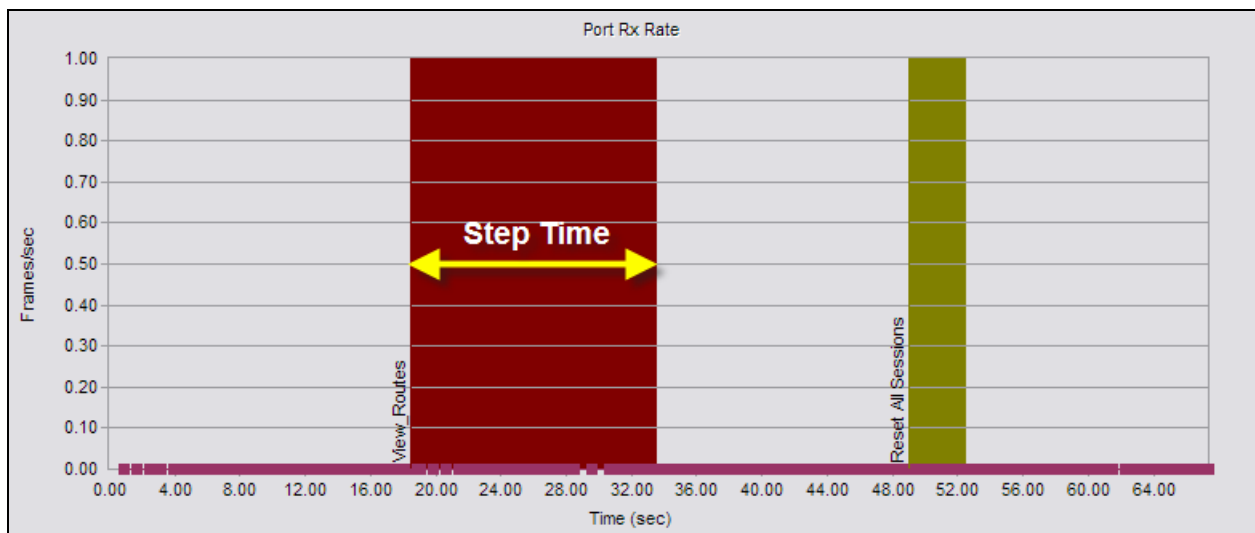
### 3.1.1 Script.ini [scripts] section

The scripts section contains two lines for each script you may wish to include in your test. You can include as many scripts as you like, each using two lines. The first line contains three fields separated by spaces. The first field is the script identifier. In this example, the script identifier is called view\_routes. The script identifier must not contain any spaces. Use underscores "\_" instead. The script identifier is what will show in the flap scheduler Add Event... dialog box.

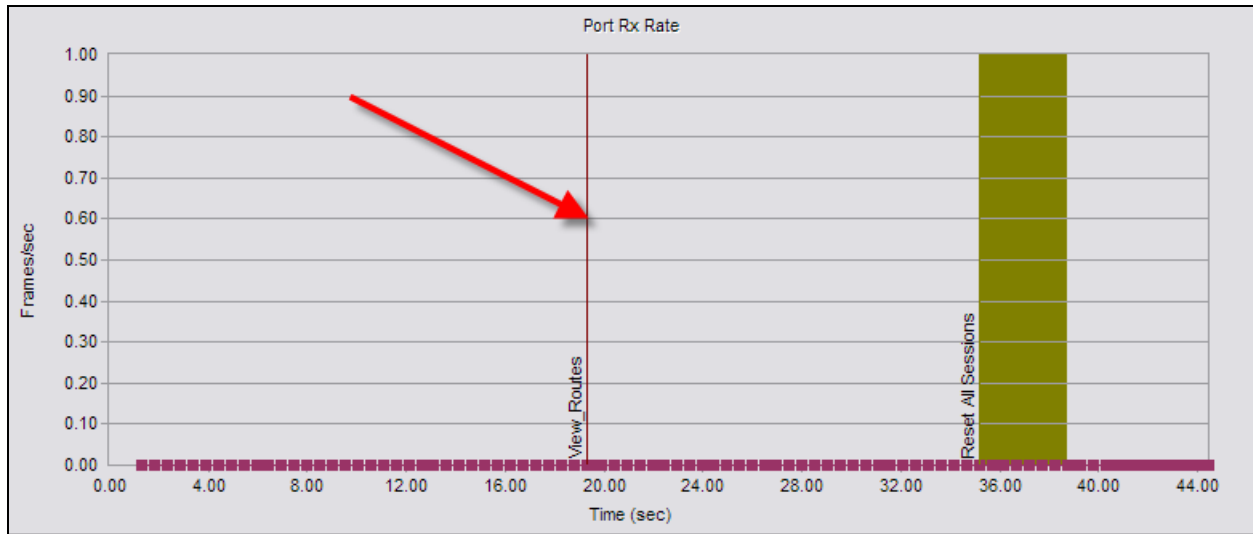


**Figure 4 – Add External Process Event**

The second and third fields after the script identifier are the blocking indicator and the timeout. The blocking indicator tells whether Spirent TestCenter should wait for the script to complete (1—blocking) before moving on to the next step in the test. A zero (0—nonblocking) tells Spirent TestCenter to continue on to the next step in the test regardless if the script has completed or not. The blocking feature can be quite handy when looking at the charts produced in the results view panes. The width of the box covering an event shows how long it took to execute the script (about 15 seconds in this example):



**Figure 5 – A Blocking Script**



**Figure 6 – A Non-blocking Script**

The non-blocking script will return immediately.

The third field of the first line of the [scripts] section is the timeout field. If and only if the script is blocking, this is the amount of time in milliseconds that Spirent TestCenter will wait for the script to complete before moving on to the next event in the flap scheduler. If a script is non-blocking the timeout field is ignored.



**Tip:** If the timeout expires on blocking scripts, the script will still continue to run in the background until it completes. When the test ends, a dialog will be presented to terminate all running processes.

The second line of the [scripts] section is the command line used to execute the script. In this example, we are using the Tcl shell to run a script called view\_routes.tcl.



**Tip:** Surround the command line arguments in quotes (“”) if your command contains spaces. Example: `tclsh84.exe "C:\White Box\config.tcl"`

### 3.1.2 Script.ini [attributes] section

The second section of the script.ini file is the [attributes] section. This describes the scripts to run at strategic points of the test. The attributes section contains flags to 1, run the appropriate script at the described time, or 0, do not run any script. You can run scripts at the beginning of the test, the beginning of the iteration—just before sending traffic, the end of iteration, and the end of the test.

```
RunScriptBeforeStartOfTest 0
RunScriptBeforeStartTraffic 0
RunScriptAfterEndOfTest 0
RunScriptAfterEachTestIteration 0
```

A script identifier is set for each of these event times:

```
StartOfTestScript script_id_one
BeforeStartTrafficScript script_id_two
EndOfTestScript script_id_three
AfterEachTestIterationScript script_id_four
```

Note that you have no control over which arguments are passed on to the script for these four events like you have from the flap scheduler.

### 3.1.3 Keeping scripts and events synchronized

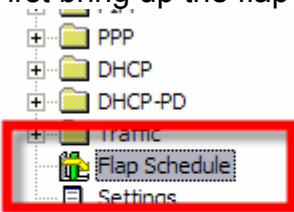
At the beginning of test, Spirent TestCenter starts from zero seconds and counts up. Each time a script is called from the flap scheduler, an argument is appended and passed to the script called `-reftime {time_in_seconds}`. This argument reflects the time from the beginning of the test up to when the script was called. This argument can be used by your script to refer to other events and the relative time between those events. When a test iteration completes, the end of iteration script gets called with `-reftime` reflecting the total time to complete an iteration. Note that this time may not be the same time as configured in the GUI, since blocking scripts can increase this time which is not counted by the GUI. The end-of-iteration script also includes `-comment "EndOfIteration,Frame Size=x,Load=y,"` denoting the frame size and the load for the iteration.



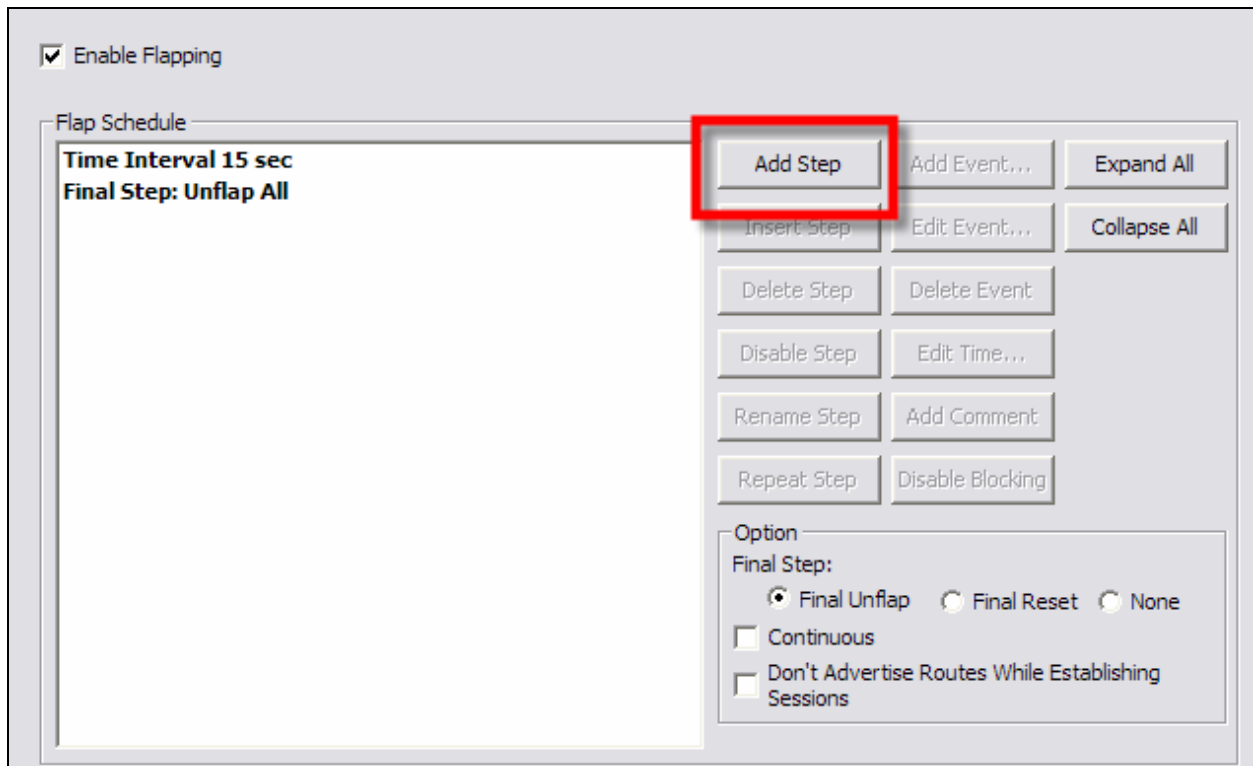
**Tip:** Use the `-reftime` feature to keep your scripts in sync with the application

## 3.2 Calling External Scripts from Spirent TestCenter

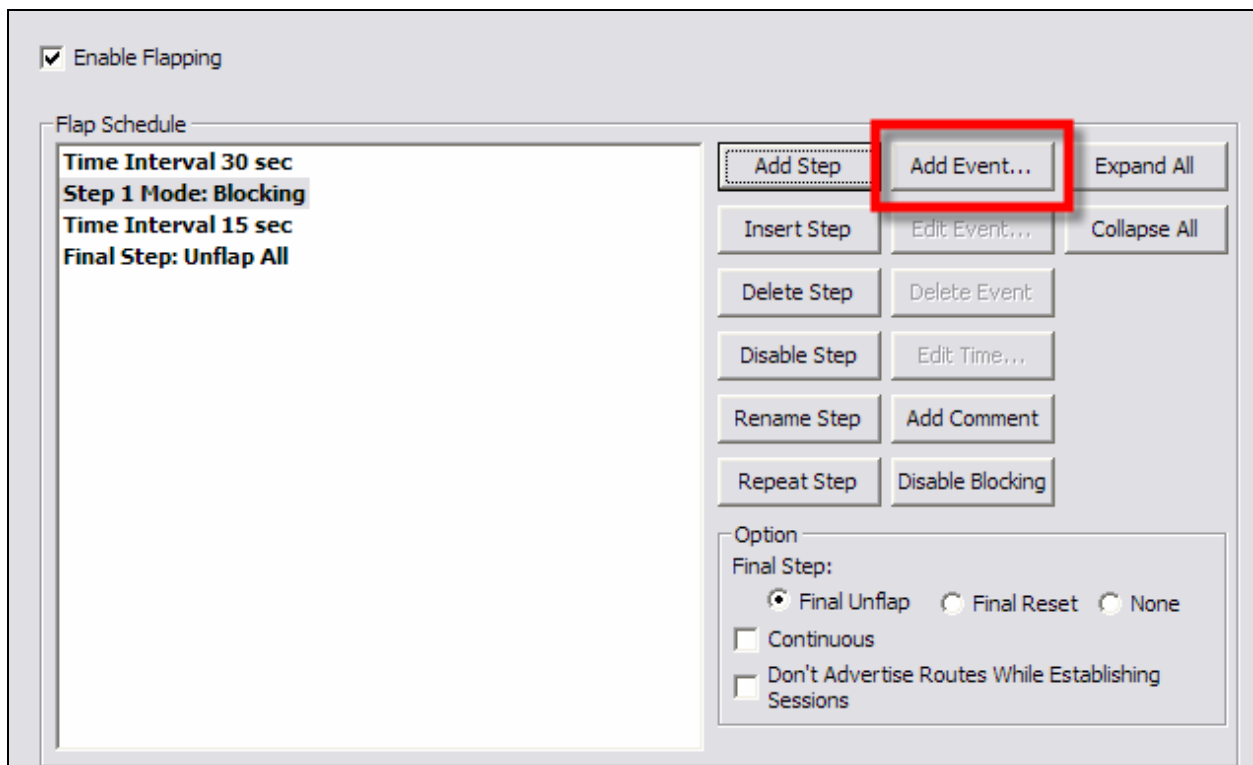
First bring up the flap scheduler, as shown:



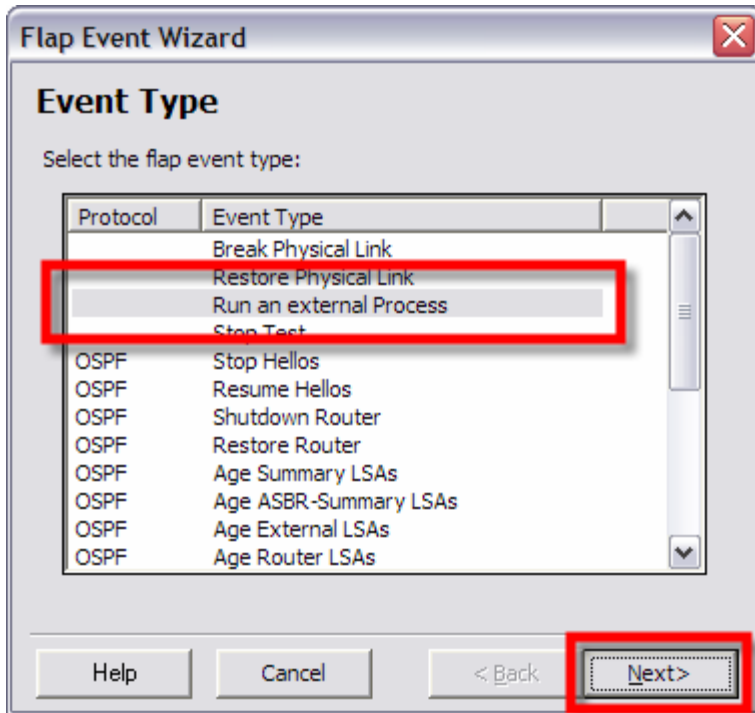
In the main flap scheduler window, add a step to the event sequence:



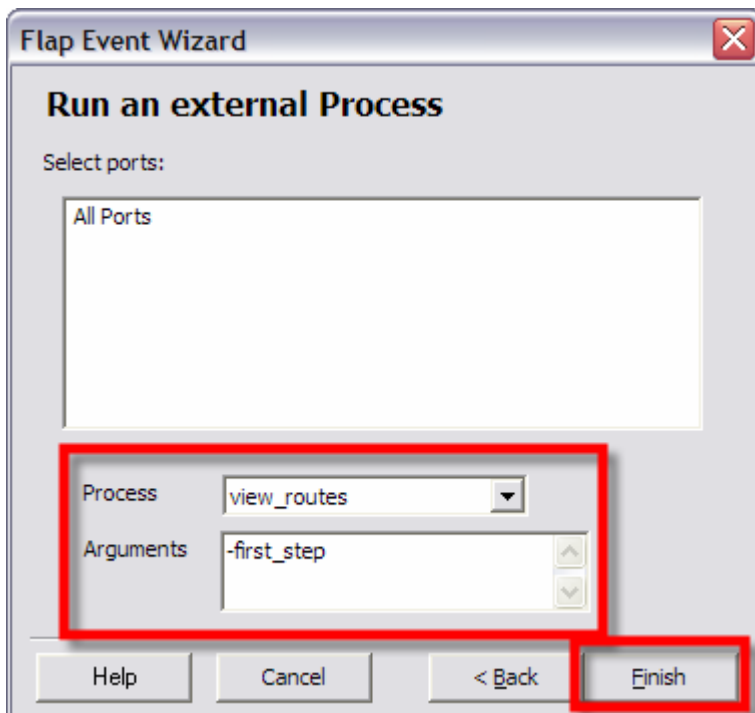
After adding a step, add an event to the step:



After adding the event, a window will appear asking for more information. Note that this event will occur 30 seconds into the test:



You want to Run an external Process. Click Next.

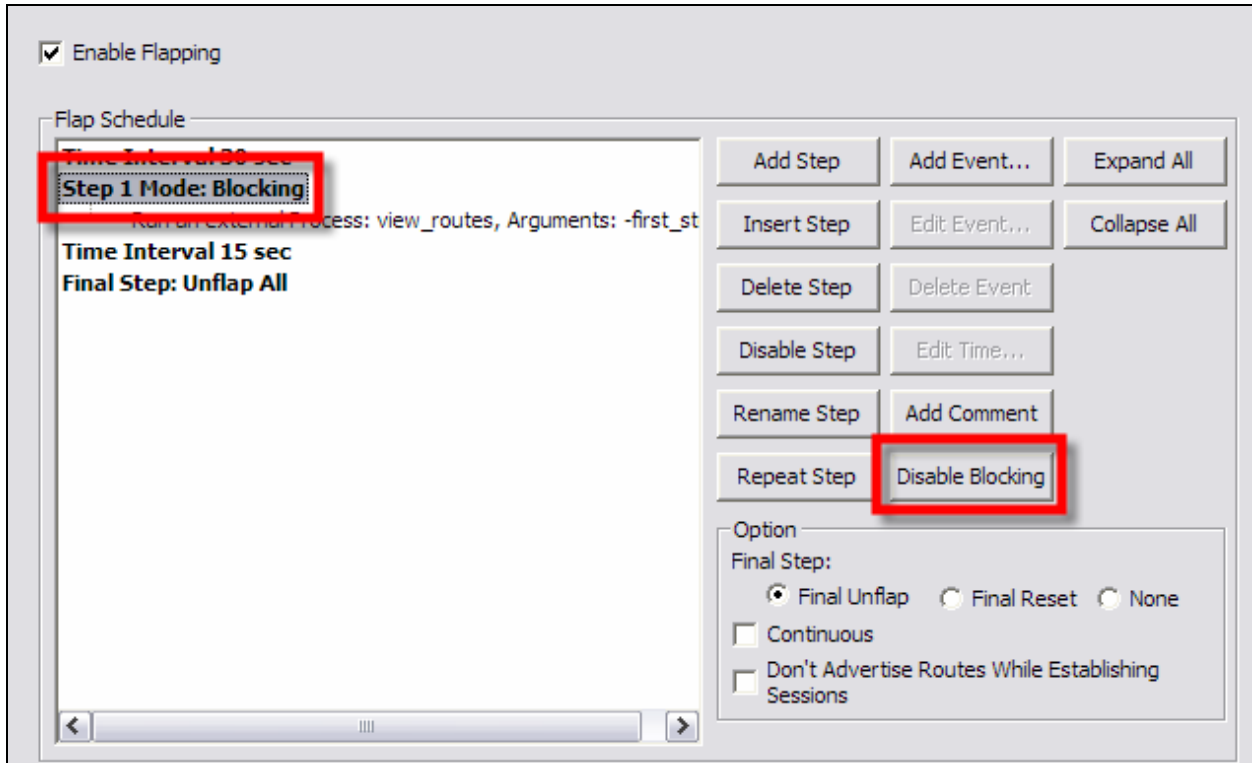


Process denotes the script identifier we previously configured in the script.ini file. We may also pass additional arguments to the view\_routes script to treat each step uniquely. Click finish and you've defined your first step in the flap scheduler. The final

command line will be: `tclsh84.exe view_routes.tcl -first_step -reftime xx.x` to be passed to your script. Xx.x will be approximately 30.0 seconds.

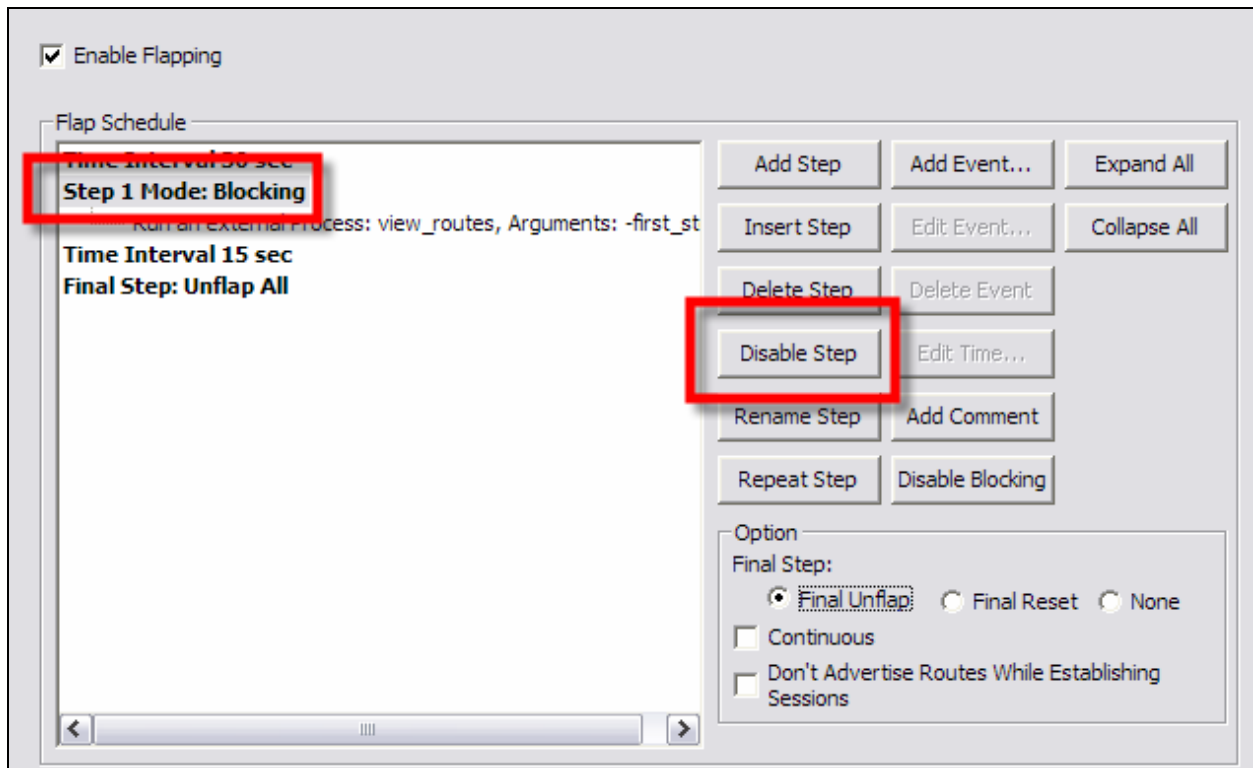
### 3.3 New Flap Scheduler Enhancements: Blocking/Nonblocking Steps

In Spirent TestCenter Release 1.20, you now have the option of blocking or non-blocking steps. Just like scripts blocking or not, so can you have a whole step with multiple events not block execution of the steps in the scheduler. Note that different types of events within the step are still blocking.



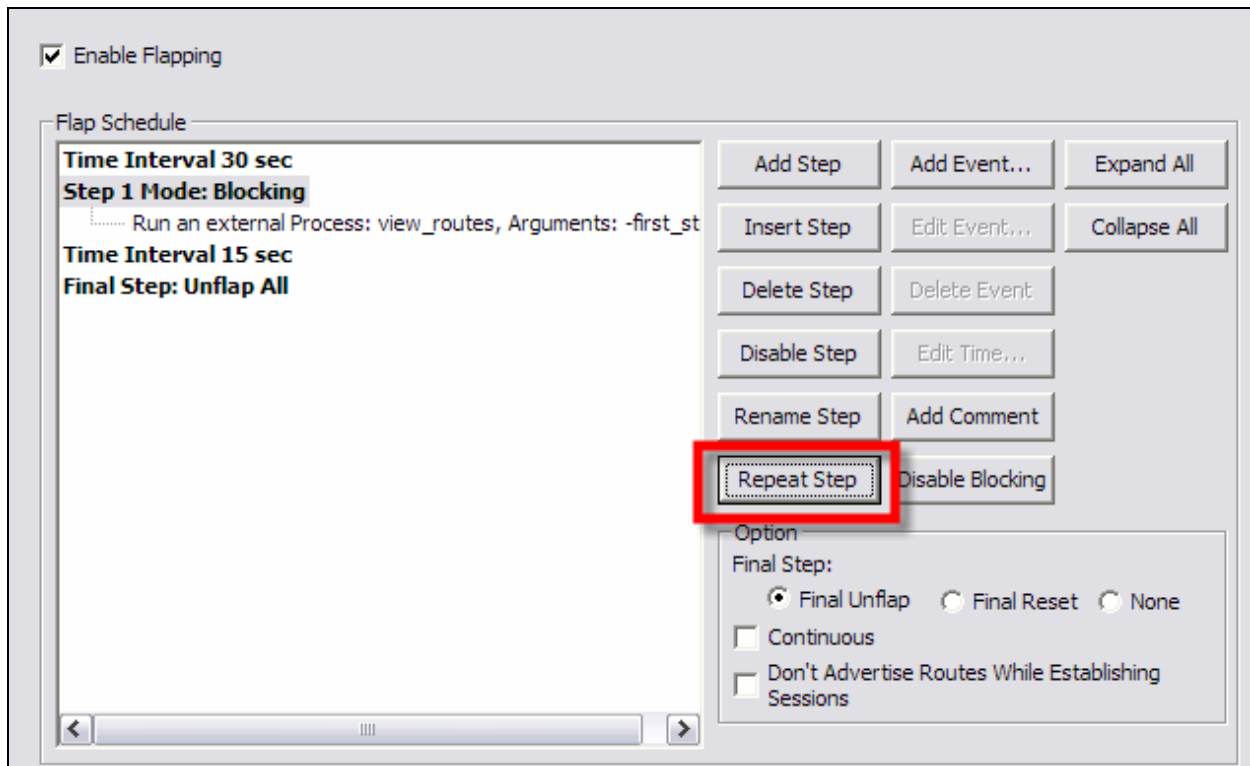
### 3.4 New Flap Scheduler Enhancements: Disabling Steps

You can now disable steps in the flap scheduler without removing them.

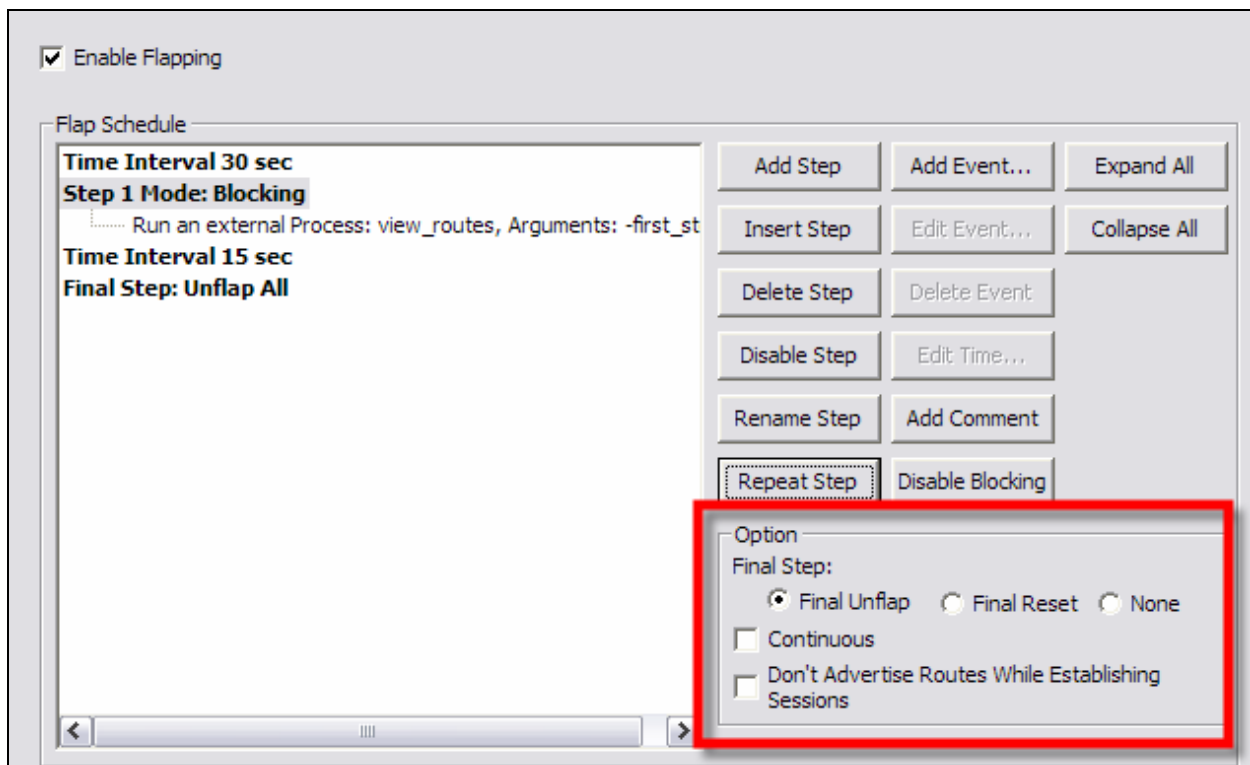


### 3.5 New Flap Scheduler Enhancements: Repeating Steps

You can also repeat a step multiple times:



### 3.6 New Flap Scheduler Enhancements: Options



The option section controls the behavior of the flap scheduler at the beginning and end of the test.

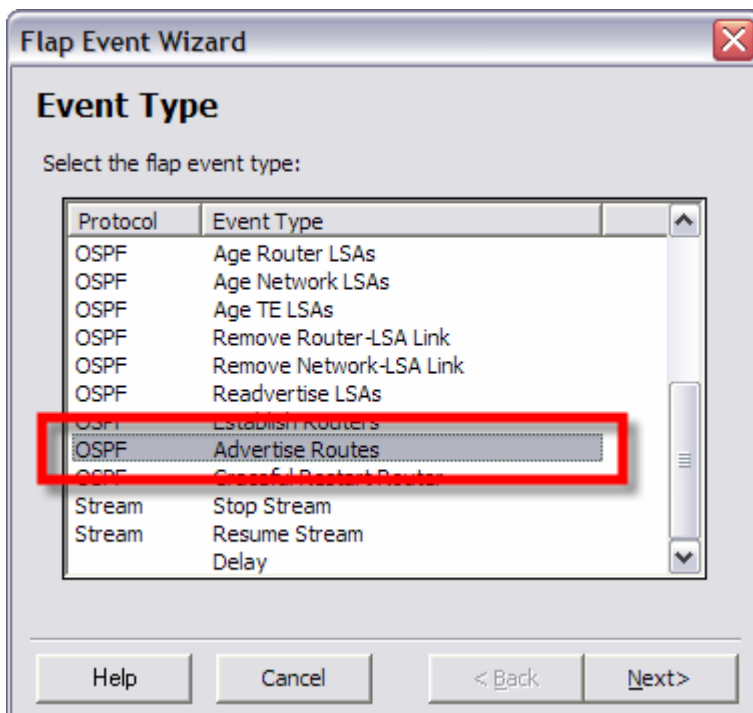
At the end of the test, you may unflap all routes, reset all sessions, or do nothing. You may also set continuous mode, which will repeat all steps until the test ends.

At the beginning of the test, you have the capability to control if routes get advertised after routing sessions/adjacencies are established.



**Tip:** Routes control is supported for BGP, OSPF, OSPFv3, RIP, IS-IS, IGMP and PIM

If not advertising routes at the start of test, you must create a step and event to advertise routes for protocols in use:



## 4. CONCLUSION

We've reached the conclusion of this application note for Spirent TestCenter. You should now have a better idea how to best utilize the tool for your white box testing needs. The application now supports custom scripts and new enhancements to tie the flap scheduler to your entire test bed. Your custom scripts may include any content you like to configure, analyze and understand your device behavior in different test scenarios.