

Tech Note

Synchronization with SwiftTest Events

SwiftTest Events is a feature that enables scenario synchronization for functional testing. For storage equipment vendors and end users, functional testing is not only used to evaluate a storage system’s ability to access and manipulate files, it also tests a system’s ability to authenticate users, navigate a file system, and perform metadata operations. With SwiftTest’s synchronization capabilities, you can test these operations as they occur simultaneously for multiple users across multiple protocols.

Testing the ability of your system to handle request collisions is notoriously difficult. This functionality is built into SwiftTest systems so you can now easily create concurrency tests.

An event is a type of SwiftTest Action that allows two SwiftTest Scenarios to synchronize execution of commands on a single file.

Here’s how it works. One scenario (or user) is programmed with a SwiftTest Wait For Event instructing it to wait, or pause execution. A second scenario is programmed to perform a SwiftTest Raise Event, or send, when it’s ready for the first scenario to continue executing.

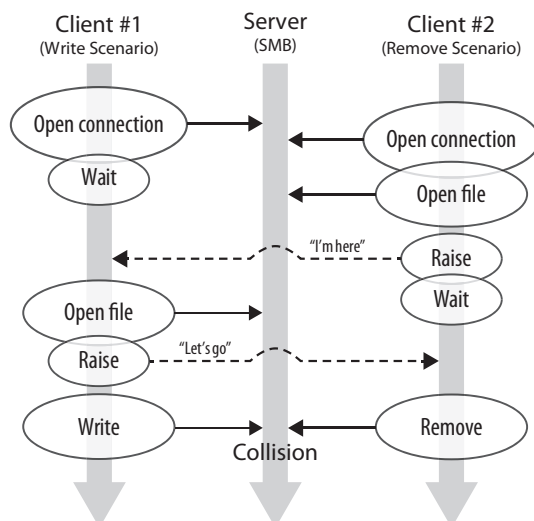
With repeated Raise and Wait For events, multiple users or scenarios are in tight communication so that they can perform operations within microseconds of each other, thereby simulating collisions.

Example: Simulation of a write/remove collision

Test a system’s ability to handle simultaneous write and remove commands using SMB.

Client #1 opens a connection and waits. Client #2 opens a connection, opens a file, raises an event that announces opening the file, and waits. Client #1 opens the same file, raises an event that announces that it, too, has opened the file, then proceeds to write. Client #2 detects the raise event, and proceeds to remove the file.

The write and remove occur within microseconds, essentially achieving collision. This is exceedingly difficult to program from scratch, but with SwiftTest Events, it’s easy.



Feature Highlights

Wait For Event Action

This action waits for the occurrence of a named event. The scenario that contains this action will wait for this event to be raised by another scenario. It will then continue executing at that point.

As with a raise event, the event key provides the means for recognition between scenarios — created by a function or may be a simple user parameter file reference. But the event key must be identical to that specified in the scenario sending the signal..

#	Protocol	Name
1	SMB	Open SMB TCP Connection
2	SMB	Negotiate
3	SMB	Session Setup
4	SMB	Tree Connect
5	SWT	Wait for Event
6	SMB	Create Or Open File
7	SWT	Raise Event
8	SMB	File Write
9	SMB	File Close
10	SMB	Tree Disconnect
11	SMB	Session Logoff
12	SMB	Close SMB TCP Connection

#	Protocol	Name
1	SMB	Open SMB TCP Connection
2	SMB	Negotiate
3	SMB	Session Setup
4	SMB	Tree Connect
5	SWT	Raise Event
6	SMB	Create Or Open File
7	SWT	Wait for Event
8	SMB	File Close
9	SMB	Tree Disconnect
10	SMB	Session Logoff
11	SMB	Close SMB TCP Connection

Raise Event Action

The purpose of this action is to send (or raise) an event of a specific name. It signals the specific event by way of a SwiftTest Event Key which is a string entered into the input field for that action — created either by a SwiftTest Function or derived from a SwiftTest User Parameter File. The action should then be inserted in the scenario at the point you require that particular event to be raised.

Scalability

The Write/Remove example is just a single pair, so consider scaling up to real-world circumstances. With SwiftTest Events run many thousands of synchronized pairs at once.

See the action lists of two users operating on the same file, and the resulting trace showing virtually undetectable delay.

The image shows a Wireshark capture of network traffic on Client Port 0 (172.17.1.45 port 0). The filter is set to 'smb'. The packet list shows a sequence of SMB operations:

No.	Time	Source	Destination	Protocol	Length	Info
33	6.033239	172.16.240.1	172.16.1.23	SMB	146	Tree Connect AndX Request, Path: \\172.16.1.23\DUMP
34	6.033495	172.16.1.23	172.16.240.1	SMB	112	Tree Connect AndX Response
35	6.033502	172.16.240.1	172.16.1.23	SMB	166	NT Create AndX Request, FID: 0x0003, Path: FILE01.TEST
36	6.033659	172.16.1.23	172.16.240.1	SMB	161	NT Create AndX Response, FID: 0x0003
81	6.033961	172.16.240.1	172.16.1.23	SMB	1382	write AndX Request, FID: 0x0003, 65500 bytes at offset 0
82	6.033970	172.16.240.2	172.16.1.23	SMB	99	Close Request, FID: 0x000d
106	6.034596	172.16.1.23	172.16.240.2	SMB	93	Close Response, FID: 0x000d
107	6.034603	172.16.240.2	172.16.1.23	SMB	93	Tree Disconnect Request
108	6.034680	172.16.1.23	172.16.240.2	SMB	93	Tree Disconnect Response

Conclusion

Executing commands concurrently helps you understand what will happen in real-world situations where timing of the commands received by a device can't be controlled.

SwiftTest Events offers concurrent command execution from two clients, which cannot be achieved through batch scripts that drive

Windows or Linux client systems. And you can complete much more concurrency-based testing, in less time, with finer resolution accuracy than with other approaches. Moreover, SwiftTest Events makes it possible to create thousands of collisions during a short test so you can discover potential issues under load.

SwiftTest Events includes additional features and advanced functionality. To get the whole story, schedule a demo: sales@swifttest.com.

