Medusa Labs Test Tools Suite

Version 7.0

User's Guide

Notice

Every effort was made to ensure that the information in this document was accurate at the time of printing. However, information is subject to change without notice, and JDSU reserves the right to provide an addendum to this document with information not available at the time that this document was created. This document was published in November 2014.

Copyright

JDS Uniphase Corporation
430 N. McCarthy Blvd.
Milpitas, CA 95035 USA

Trademarks

Support and Maintenance

Technical Support:

Telephone:	US Toll Free:	1 855 ASK-JDSU, select 3, 3, 3 1 301 353-1560, select 3
E-mail:		E-mail: techsupport-snt@jdsu.com

Contacting Medusa Labs

You can contact Medusa Labs Monday through Friday, 8 a.m. to 5 p.m. Central Time.

Medusa Labs, 800 Paloma Drive, Suite 130, Round Rock, TX 78665

Phone: (512) 670-7300	E-mail: techsupport-medusa@jdsu.com
Fax: (512) 670-5078	Website: www.jdsu.com/go/medusa

Terms and Conditions

Specifications, terms, and conditions are subject to change without notice. The provision of hardware, services, and/or software are subject to JDSU's standard terms and conditions, available at www.jdsu.com/terms.

Contents

About this Guide

Congratulations on your purchase of the Medusa Labs Test Tools Suite.

This guide describes the Medusa Labs Test Tools (MLTT) features and provides information about how to use the tools to test your devices.

For information on installing Medusa Labs Test Tools and the License Manager, see the *Medusa Labs Test Tools Suite Installation Guide*.

What this Guide Contains

This guide contains the following chapters:

Chapter 1, "About Medusa Labs Test Tools" describes the MLTT capabilities. It also provide examples of test configurations to use with MLTT and includes system requirements and information about licensing.

Chapter 2, "Using the Graphical User Interface" describes the graphical user interface and menu items.

Chapter 3, "Using the Configuration Editors" describes each of the configuration editors of the graphical user interface.

Chapter 4, "Using the Command Line Switches" describes the test commands you use to stress your devices.

Chapter 5, "Logging and Output" describes the logs that are generated when you use MLTT.

Chapter 6, "Data Pattern Reference" discusses using data patterns in your tests and how you can customize them.

Chapter 7, "Catapult Test Tool Automation" describes how to use the discovery tool that also includes scripting features to automate tests you want to run.

Appendix A, "Data Pattern Numbers" lists the numbers you use to call data patterns in your test.

Appendix B, "Test Guidelines and Examples" contains examples of how to use the switches and descriptions of tests you can run.

Appendix C, "Debug Example," describes how MLTT is used in a practical scenario and details the process of tracking errors.

Appendix D, "I/O Signatures" describes how to use I/O Signatures.

Appendix E, "Exit Codes" provides the exit codes and their descriptions.

Appendix F, “Architecture Bandwidths” lists the bandwidths for various architectures.

Conventions

The following conventions are used in this guide.

Message Formats

This guide uses the following format to highlight special messages:

Note: This format is used to highlight information of importance or special interest.

Important: This format is used to highlight information that you should know.

Caution: This format is used to highlight information that will help you prevent equipment failure or loss of data.

Warning: This format is used to highlight material involving possibility of injury or equipment damage.

Typographical Conventions

This guide uses the following typographical conventions:

bold sans serif	Commands
italics	Directory names, book titles, named key, for example the Enter key.
`courier font`	Screen text, user-typed command-line entries.
`courier italics`	user-supplied variable, argument

Chapter 1

About Medusa Labs Test Tools

In this chapter:

What's New in this Medusa Labs Test Tools Version

The Medusa Labs Test Tools (MLTT) Suite version 7.0 has the following new features:

- **IOMeter** allows IOMeter Configuration Files (.icf and .txt files) to be imported into MLTT as test plans. Refer to "Import Test Plans..." on page 25 for more information.
- **SSD Secure Erase** erases the data on a Solid State Drive (SSD) leaving it in a *clean* state.
  - Refer to "SSD Secure Erase Configuration Editor" on page 119 for information about using this feature with the graphical user interface (GUI).
  - Refer to "--secure-erase Erase the Target Device and Exit" on page 145 for information about using this feature with the command line inputs.
- **SSD Trim** erases specified data blocks. It may be run as a target Solid State Drive (SSD) pre-conditioning step before running I/O tests.
  - Refer to "SSD Trim Configuration Editor" on page 121 for information about using this feature with the graphical user interface (GUI).
  - Refer to "--trim Send Trim to Target" on page 147 for information about using this feature with the command line inputs.
- **Steady State** determines the steady state for a target across five consecutive test runs. When steady state is achieved, the test plan will be stopped when the current test iteration completes and if the test plan is part of a planning group, the next test plan in the group is started.
  - For information about using this feature with the graphical user interface (GUI), refer to "Steady State" on page 68 for Custom configurations or refer to "Steady State" on page 92 for Performance configurations.
  - Refer to "--steady-state Determine Steady State" on page 142 for information about using this feature with the command line inputs.
- **Latency Histogram** collects and displays latency histogram data per target using user-specified bins which are sorted by the magnitude.
  - For information about using this feature with the graphical user interface (GUI), refer to page 68 for configuration information and refer to "Latency Histogram Tab" on page 60 for display information.
  - Refer to "--latency-histogram Collect Latency Histogram" on page 144 for information about using this feature with the command line inputs.
- **S.M.A.R.T Monitoring** retrieves Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) attributes and status from target devices and logs them.
  - Refer to "--smart S.M.A.R.T Monitoring" on page 169 for information about using this feature with the command line inputs.

What Medusa Labs Test Tools Does

The Medusa Labs Test Tools (MLTT) Suite performs data integrity testing, signal aggravation, and enterprise application simulation.

Medusa Labs develops test tools that meet the extreme demands of enterprise test and development engineers. The superiority of these tools is due to several factors, including:

- **The Tools are fast and efficient.** In many baseline evaluations, we find that our tools are generally faster and more processor efficient than any other test applications in the industry. In many cases, we are able to achieve throughput greater than the fastest industry-standard benchmarks. What's even more impressive is that on many high-end systems, we are able to write, read, and compare data faster than many benchmarks performing the same I/O without data integrity checking. When the technology allows for full duplex, we find that our test tools can many times achieve 100% greater throughput than other available benchmarks or tools, as the majority of them were developed with half duplex (or bus) operations in mind.
- **The Tools are precise and highly specific.** Many test tools used during development attempt to stress the aggregate system. Medusa Labs designs our test tools in a manner that allows them to stress specific and unique areas of an enterprise system. Although our tools are designed for specific uses, you can easily set them up to stress the aggregate systems or set them up for full scale enterprise testing.
- **The Tools are designed with debug and analysis in mind.** Finding bugs is easy. Characterizing their behavior and eventual root cause analysis can be tricky. We have designed our tools to send out unique data patterns (to trigger analyzers) when they discover a data anomaly, such as data corruption or data loss. We also insert identifying data values into our data patterns that allow the test engineer to better determine and track the client and/or thread that potentially is the cause or catalyst for the error condition.
- **The Tools contain specific data patterns and routines that best stress various architectures.** As a number of our test services customers have seen, these data patterns and routines can aggravate or be a catalyst for quicker reproduction of issues. From our experiences, we have found that a substantial number of tested components will readily show certain failure types when subjected to only data pattern specific high stress testing.
- **The code is portable.** Our command line tools are supported on Windows®[1], Linux®[2], HP-UX®[3], and Solaris (SPARC-based and X86-based) platforms. A consistent user interface makes it easy for test engineers to move between platforms.
- **The Tools are designed to bypass multiple layers of the operating systems.** In order to fully stress the hardware under test, our tools have settings that allow for partial or complete bypassing of several layers of the OS that inhibit sustained high stress testing. Tools can be executed with switches to request cached or non-cached I/O. Target access can be directed at file systems, logical devices, or physical devices to enable the test engineer to drill down to the desired layers.

[1] Windows is a registered trademarks of Microsoft Corporation in the United States and/or other countries.
[2] Linux is a registered trademark of Linus Torvalds.
[3] HP-UX is a registered trademark of Hewlett-Packard Company.

How Medusa Labs Test Tools Works

Our test tools are user-mode command-line applications that run on a host system. At the simplest level, our test tools operate in an initiator-target fashion. The host system acts as an initiator and the target can be any storage device internal or external to the host system. With our test tools, the host system becomes a precision traffic generator using real-world application data. Because the tools are command-line based, they are ideal for setting up scripted test runs. A Graphical User Interface (GUI) is also available on Windows platforms that duplicate the command-line options.

Pain and Maim Test Tools

Pain and **Maim** are the currently available I/O test tools in the Medusa Labs Test Tools Suite.

Pain is a synchronous I/O tool that is designed to issue a single pending I/O per worker thread.

Maim is an asynchronous I/O tool that is designed to issue multiple pending I/Os per worker thread.

Table 1 shows a comparison of these tools.

Table 1: Test Tool Comparison

Pain	Maim
Synchronous I/O	Asynchronous I/O
Single pending I/O per worker thread	Multiple pending I/Os in multiple worker threads
Separate file or device offset range for each thread	Single file or device offset range.
Static queue depth	Static or fluctuating (bursting) queue depth
Supports a memory only mode	Target device access only
Excellent full system and target stress testing	Focused target testing
High thread counts will create processor overhead	Extremely processor efficient

Sock Test Tool

Sock is a TCP network I/O test tool where each worker thread simulates a client/server connection performing synchronous I/O to exchange data. Sock can be used very much like Pain (e.g. read-only, write-only, write-read with data comparison, etc.) or in transaction mode with various I/O profile settings to simulate I/O generation of network applications such as HTTP Web transactions.

Catapult Test Tool Automation

Catapult is the target discovery tool included with the test tool suite that acts as a shell for the I/O tools. You use Catapult to discover targets available to the host system and pass these targets to the other test tools for I/O testing. There are also features in Catapult that facilitate test scripting and automation. Refer to Chapter 7, “Catapult Test Tool Automation” for more information about this tool.

FindLBA Utility

FindLBA is a utility application you can use when debugging data corruption issues in tests on file systems or logical devices. It is useful in cases where the logical block address (LBA) reported in the I/O tool error logs is not accurate because the tools are not directly referencing areas of the physical media. You can use FindLBA in conjunction with a protocol analyzer to identify the actual LBA corresponding to a file offset reported by the test tools. FindLBA sends a “ping” of consecutive reads to a specified offset, which you can identify in a protocol trace. FindLBA is most useful when you need help finding I/O commands that resulted in data corruption in a protocol trace capture. Refer to “Using the FindLBA Utility” on page 256 for examples of using this utility.

GetKey Utility

GetKey is a utility application used for remote license checkouts. A system with network access to a license server can perform a license checkout for another system that does not have network access. This utility is particularly useful for temporarily using MLTT at an off site location.

Medusa Agent

Medusa Agent is a Windows service or a Unix daemon process of MLTT that provides the following functions.

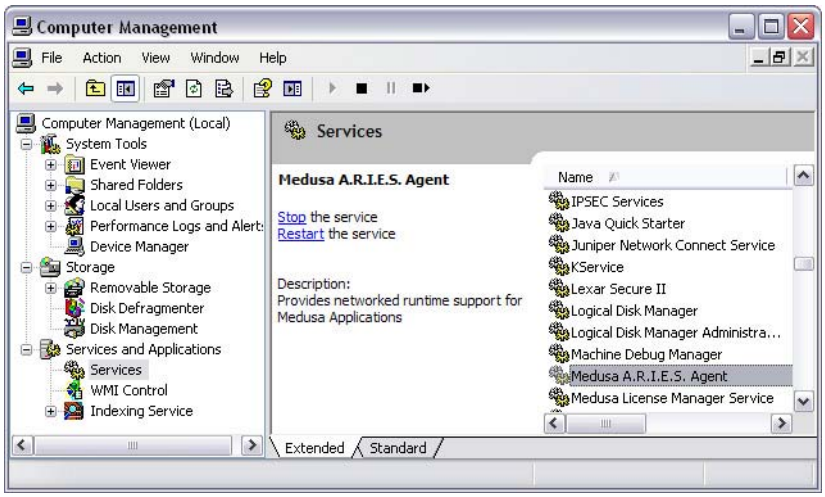
- Local license client
- Discovering other systems running MLTT in the network
- Mediate remote execution of MLTT

The agent uses TCP and UDP to communicate with other systems. The service is installed and configured during Medusa Labs Test Tools installation, and direct user interaction with the agent process is usually not necessary.

However, in case the service needs to be stopped or restarted manually:

On Windows, this can be done in the service control management GUI. As shown in Figure 1, the agent is registered with "Medusa A.R.I.E.S. Agent" as the service name.

Figure 1: Service Control Management Screen



From the command line:

On Windows: "net stop maagent" to stop the service
"net start maagent" to start the service

On Linux: "service maagent stop" to stop the service
"service maagent start" to start the service

On Solaris: "/etc/init.d/maagent stop" to stop the service
"/etc/init.d/maagent start" to start the service

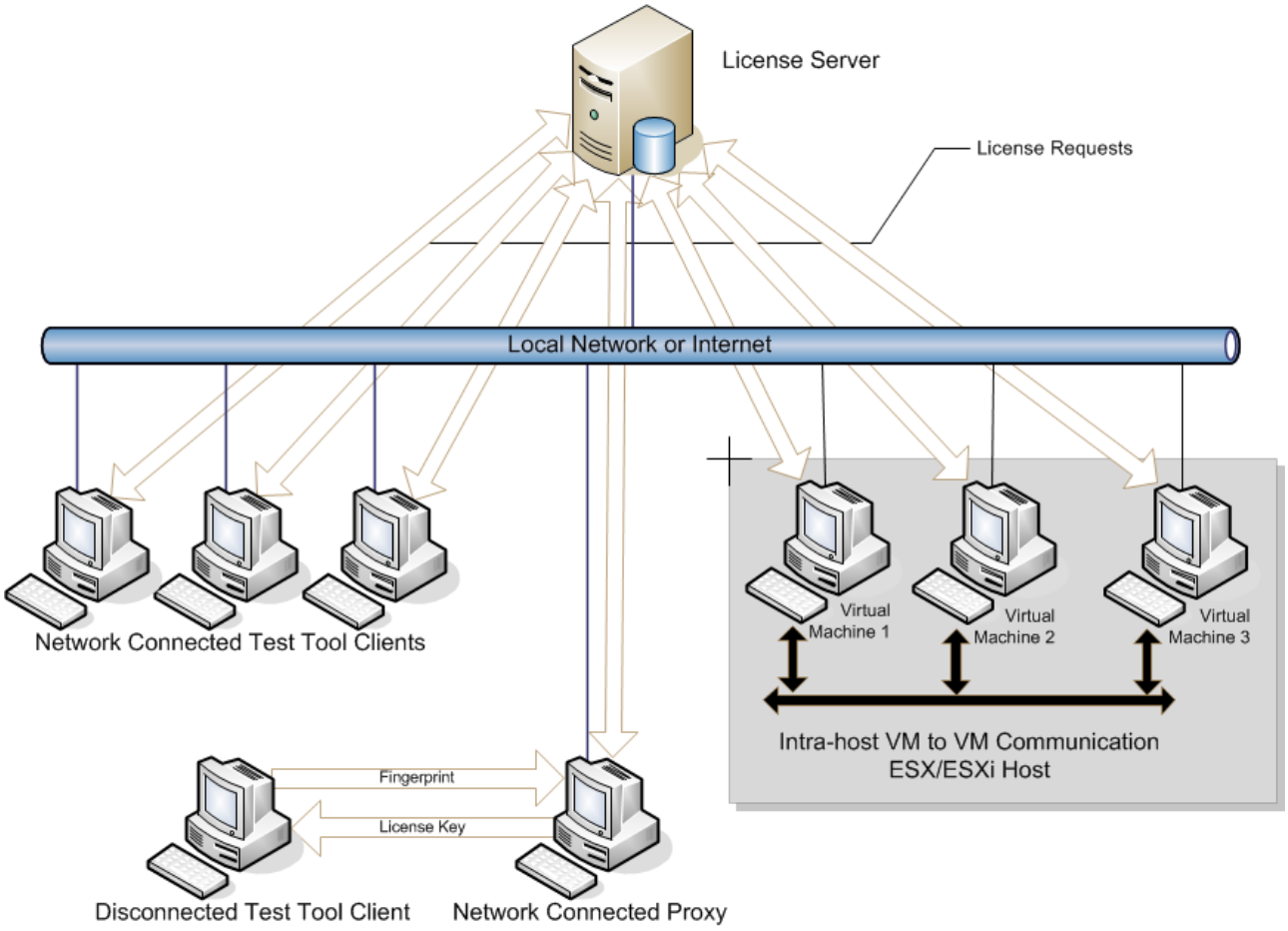
On HP-UX: "/sbin/init.d/maagent stop" to stop the service
"/sbin/init.d/maagent start" to start the service

Licensing

Medusa Labs Test Tools (MLTT) are licensed on a “per seat” subscription basis. This means that the tools are licensed for a period of time (usually one year). A certain number of “seats” are licensed under the subscription. A seat is any client system that is currently running MLTT. You can use the license seats on any system, but concurrent usage is limited to the number of seats purchased. This methodology allows you to use the tools where they are needed, without being restricted to particular systems. In addition to Regular Licensing that is described in this section, MLTT supports Virtual Machine Licensing that is described in depth in “Virtual Machine Licensing” on page 10.

License usage is regulated by a license server that issues license keys in response to checkout requests by client systems, up to the number of seats purchased. Refer to Figure 2. The license server software is provided for on-site installation of the license server. Medusa Labs provides a hardware security dongle to enable the license server installation.

Figure 2: Medusa Labs Test Tools License Model



You can check out license keys from the license server for any client system. The issued key will work only for the client system that checked it out. A license key is time limited and the duration of the checkout is configurable. A license seat is consumed on the license server when you perform a checkout. The license seat remains allocated to the system that checked out the key until the checkout time limit is exceeded or you perform a check-in.

All licensing information is stored on the client system and no further contact with the license server is required during the checkout period. Checked out keys expire when the key's time limit is exceeded; the license seats automatically become available on the license server. At this point, the tools would need to be able to checkout a new license key to replace the expired key if further use of the tools is required. If use of the tools on a system is completed before the time limit is reached, a check-in may be performed to return the key to the server and make the license seat available for other systems.

In most cases, the systems running the tools will checkout a license key from the license server directly over a network. However, a networked system can perform a checkout on behalf of a system that is not connected to a network or that does not have network access to the license server. This is called a remote checkout and it is accomplished with the GetKey utility. A remote checkout requires a machine lock file (fingerprint) created by the tools be transferred to the networked system performing the remote checkout. This lock file is used by GetKey to request a license authorization code from the license server.

Only the I/O generating test tools (Pain, Maim, and Sock) require a license key. All the test tools running on a client system use the same key.

Licensing Requirements

To checkout a license and run MLTT, your client systems must meet the following minimum requirements.

- To check out a license directly from the license server, the client system must be attached to a network with access to the server. TCP/IP must be properly configured on the client system. It is important to make certain that UNIX systems setup as DHCP clients are able to resolve their own host names.
- The time on the client system must be accurate. Because the license checkouts are time limited, the times on the client and the license server need to be reasonably close. A license checkout may fail if the time discrepancy between the client and server is too great. The client and server can reside in different time zones, as long as the local time is accurate for both systems.

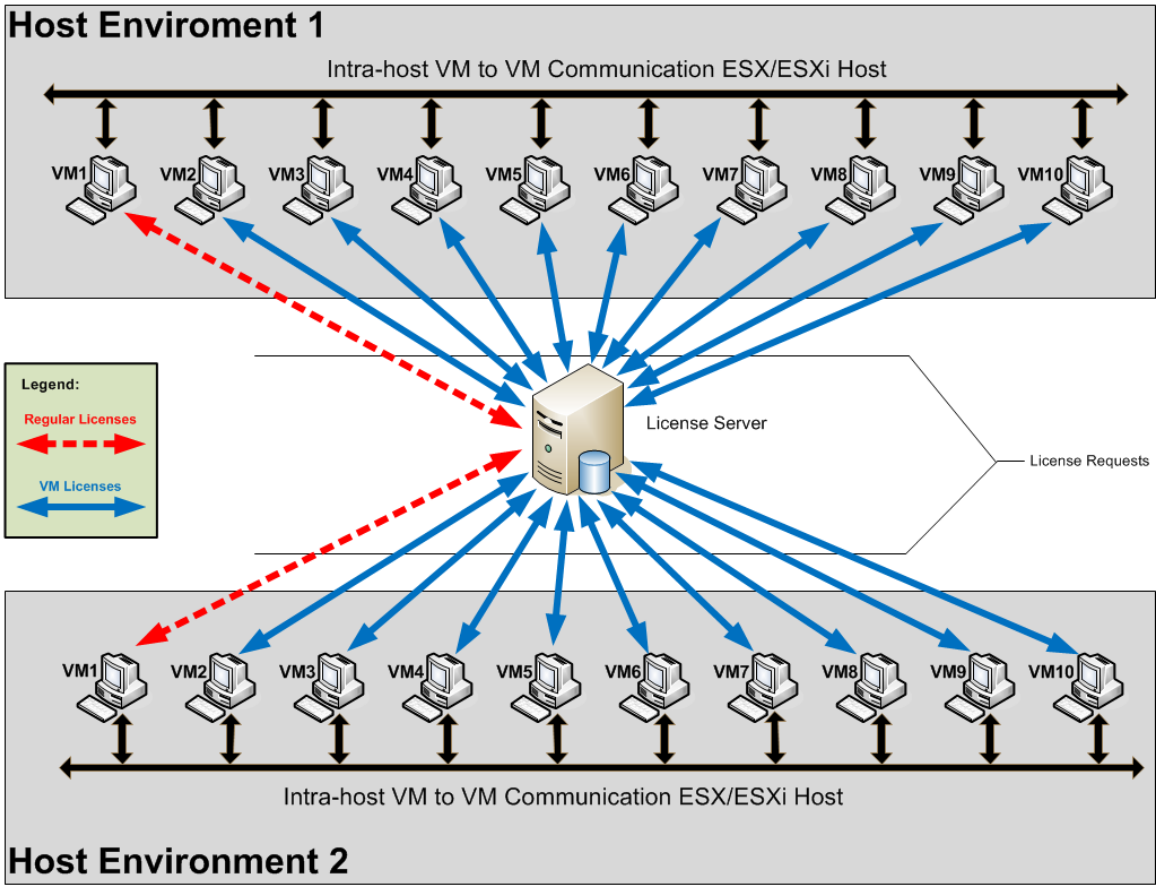
Virtual Machine Licensing

A Virtual Machine (VM) License is a new license type added to Medusa License Management (MLM) Server 1.2.1 and Medusa Labs Test Tools 6.0.0. The optional VM licenses provide the same capability as regular licenses but are more cost effective on per seat basis. However, VM licenses can only be granted to supported and recognized virtual machines being hosted on VMware[1] ESX and VMware ESXi servers.

VM licenses are considered valid only when at least one VM on the host environment has regular license checked out. For example, an MLTT deployment site has two VMware ESX/ESXi servers with each server hosting ten VMs. Each ESX/ESXi server (with its ten VMs) is considered a “host environment”. With the new VM licensing option, you can purchase the less expensive VM licenses. However, because at least one regular license per host environment is required, at least two regular licenses (one checked out to a VM on each server) are needed for this scenario. Refer to Figure 3.

[1] VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions.

Figure 3: VM Licensing Example Diagram



To get all 20 VMs licensed, one VM in each host environment checks out and uses a regular license while the other nine VMs in the host environment can use VM licenses. The VM designated to use the regular license is the “VM license proxy” of that host environment.

VM Licensing Client Requirements

Virtual machines must meet the following requirements to use VM licensing:

- The only “supported, recognized” virtual machines are hosted on VMware ESX/ESXi servers.
- The guest OS must be Windows or Linux.
- The guest OS must have the VMware Guest Tools installed.
- Unrestricted Virtual Machine Communication Interface (VMCI) must be enabled in the host environment.

Deploying VM Licensing

The following is an overview of the basic steps required to deploy VM licenses within your VM host environments. The installation steps are described in detail in the *Medusa Labs Test Tools Suite Installation Guide*.

1. Install MLM Server 1.2.1 (or newer) on the license server.
2. Purchase a number of VM licenses along with the needed number of regular licenses.

3. Install the purchased licenses into the MLM Server's HASP dongle using the same procedure outlined in the installation guide.
4. Once installed, running **mlmadmin -reportserverstate** displays two products (`MedusaLabs Test Tools version 1` and `MedusaLabs Test Tools version 1 for VM`) and the number of seats purchased for each tool.
5. Install Medusa Labs Test Tools 6.0 (or newer) on the VMs.
6. For each host environment, designate one of the VMs running Medusa Labs Test Tools 7.0 as the "VM license proxy" by executing **mlmadmin -vmproxy 1** in the designated VM.

   To revoke the "VM license proxy" designation for a VM, execute **mlmadmin -vmproxy 0** command.

Remote Checkout

When a client system is unable to contact a license server and perform a checkout directly, you can use the remote checkout approach. You can also use remote checkouts to temporarily share MLTT with a third party for reproducing test scenarios. Any Windows system with network connectivity to the license server can perform a remote checkout with the GetKey utility.

1. Run one of the I/O Test Tools (**Pain** or **Maim**) on the client system.

   When a checkout attempt fails, the tools automatically generate a system fingerprint file in the config directory. The file is named after the system host name, with a `.dat` extension, for example: `myhost.dat`.
2. Transport the `.dat` file from the client system to the networked system running **GetKey**, with access to the license server.

   You can do this using any available method, including: floppy, USB flash drive, peer to peer network, etc. If you are an off-site user, you can also e-mail this file for checkout from another location and the license key can be e-mailed back.
3. Run the **GetKey** utility on the networked system to perform a checkout for the client system. The path to the configuration file is passed to **GetKey** with the `-f` switch. You specify the number of days for the checkout with the `-z` switch (for example: `getkey -fmyhost.dat -z3`).
4. **GetKey** will contact the license server and request a license checkout. If successful, it will create a file in the current directory with the same name as the fingerprint file, with a `.lic` extension (for example, myhost.lic.)
5. You must take this file back to the client system to install the license code.
6. On the client system, run one of the I/O tools (Pain or Maim) with the license switch used to install the authorization code. The syntax of this switch is: `-Z#file_name`, where `file_name` is the location and name of the authorization code file created with **GetKey**.

   Example:

   ```
   Pain -Z#c:\temp\myhost.lic
   ```

   Use the `-Z` switch to find a machine lock file (fingerprint) and to import license file back in. This is used during a remote check out.
7. The tool will install the license and display the checkout time available. You can now use any I/O tool for the checkout duration.

To return a remote checkout:

1. Run getkey `-r` on the remote system. This will deactivate the license and create a license return file named after the remote system. Ex: `remote_system_name.ret`.
2. Take the .ret file to any system with network connectivity to the license server. Run getkey -rremote_system_name.ret to return the license seat to the license server.

Migrating the MLM License Server

To migrate the MLM License Server to a different machine requires the following backup/restore procedure in order to avoid having a corrupted server state. The procedure must be performed directly at the old and new license server systems because it requires physically removing and plugging in the USB Key.

At the current license server system:

1. Open a command window.
2. Stop the MLM License Server by running `"net stop mlms"`.
3. Unplug the MLM License Server USB Key

   WARNING: Do not plug the USB Key back in on any system until this backup/restore procedure is completed.
4. Backup the MLM License Server state by running **`mlmadmin -backup`**. This will create the `mlms.backup` file.
5. Copy it to the new system.

At the new license server system:

1. Install the MLMS software on the new system.

   WARNING: Do not plug the USB Key back in on any system until this backup/restore procedure is completed.
2. Open a command window.
3. Use `cd` to open the directory where the `mlms.backup` file from the old system was copied.
4. Restore the MLM License Server state by running `mlmadmin -restore`.
5. Plug in the MLM License Server USB Key.
6. Start the MLMS service in the new system by `running net start mlms`.
7. Make sure the Medusa Labs Test Tools client machines are configured to use the new MLM License Server.

System Requirements

The Medusa Labs Test Tools (MLTT) are designed to utilize system resources as efficiently as possible. However, performance and stress testing is by nature resource intensive. Specific system requirements will vary with the architectures under test. Generally speaking, in order to achieve full duplex wire speed levels of throughput with data integrity checking on a topologies such as Fibre Channel and Gigabit Ethernet an enterprise class system is desired.

MLTT will take advantage of multiple processors.

System Limitations

Architecturally, the tools are capable of generating tremendous I/O loads through high queue depths, large buffer sizes, and various optimizations. However, the host system hardware and operating system is often the limiting factor in what I/O parameters you can specify for a test and what the actual throughput to the target is. It is important that you take into account the effects of the MLTT command line switches on your system resources.

Memory Utilization

MLTT can demand a tremendous amount of system memory, depending on the values indicated for buffer size and thread count or queue depth. By design, the tools allocate three memory buffers for each I/O. There is a buffer for forward write data, reverse write data, and read data. This means that for each worker thread or queued I/O, buffer memory equivalent to three times the requested buffer size is allocated. You can use the following equation to determine memory requirements based on buffer size and thread count or queue depth:

```
(Buffer Size (-b#) x 3) x (Thread Count (-t#) or Queue Depth (-Q#))
```

For example:

```
pain -t10 -b512k
```

Or

```
maim -Q10 -b512k
```

These command lines both result in a buffer allocation of 15MB (512k x 3 = 1.5MB; 1.5MB x 10 = 15MB). This is in addition to the base memory footprint of the tools, which is 5 to 10MB, depending on the specific tool. It is especially important to keep memory allocations in mind when running multiple instances of the tools to a number of targets.

Processor Utilization

MLTT is designed to keep processor utilization as low as possible for most I/O testing. However, this is another area where the number of pending I/Os and the size of the I/O buffers can cause the tools to hit a system bottleneck. Data integrity checking generates a load on the processors that increases with the buffer size. With data comparisons enabled, every byte of read data is compared against write data in the default comparison mode. Extremely large buffer sizes require substantial processor time to walk through each buffer. I/O operations (IOPS) focused performance testing can also place a tremendous load on the processors, as higher queue depths with smaller I/O requests sizes are typically used. The increased frequency of context switching required by this type of testing results in greater processor utilization.

Firewalls

In Medusa Labs Test Tools (MLTT) 6.0.1 or later, MLTT dynamically manages the firewall rules. This means that under most situations, it is no longer necessary to disable the firewall for MLTT.

However, sometimes this automatic firewall management may not be possible due to site policies or specific user configurations. In such cases, you will need to provide manual firewall configurations if you want to use MLTT's networking features, such as remote testing.

In Windows, the preferred method is to add the MLTT executable files to the firewall exceptions. The executable files to be added to the exception list to allow incoming network connections are the "<install_dir>\Test Tools\bin*.exe" files.

In Unix, application-based rules are generally not available and the firewall rules must be port-based. Because MLTT uses dynamic ports, it is difficult to define a port-based rule. Therefore, if the new dynamic rule management feature of MLTT is not possible on your system, the firewall may need to be deactivated.

However, under typical default conditions, the new dynamic firewall rule management feature should be sufficient for hands-off operation.

Operating System Restrictions

The host operating system might impose restrictions such as limited thread count, maximum queue depth, concurrent file handles, and others. MLTT is designed to return operating system specific error messages whenever possible to assist with the debug of OS-related error conditions. You should also take into account the OS handling of I/O requests. Some test cases, particularly those involving file systems, might yield incorrect performance results due to OS caching.

Important: You should understand that operating systems or drivers typically break apart large I/O requests into several smaller ones. A large specified block size does not necessarily mean that the target will receive the entire I/O size at once.

The reverse is also true. Often, device drivers will coalesce small I/Os into one larger I/O.

You can use a protocol analyzer to verify the I/O transfer size.

Windows User Account Control (UAC) Restrictions

Running MLTT can be affected by the Windows User Account Control (UAC) state. The following table summarizes how the UAC state will affect MLTT operation:

Table 2: Pros and Cons of Using UAC with MLTT

UAC ON	**UAC ON, run MLTT "Run as administrator..."**	**UAC OFF, run MLTT as normal (no-elevated) admin user**
Pros: • UAC is on • Windows 8 Metro Live Tiles works correctly • Catapult can see mapped network shares • GUI can see mapped network shares	**Pros:** • UAC is on • Windows 8 Metro Live Tiles works correctly • Catapult can determine local physical drive attributes • GUI can determine local physical drive attributes • Pain/Maim can run to local physical drives	**Pros:** • Catapult can determine local physical drive attributes • GUI can determine local physical drive attributes • Pain/Maim can run to local physical drives • Catapult can see mapped network shares • GUI can see local network shares
Cons: • Catapult cannot determine local physical drive attributes • GUI cannot determine local physical drive attributes • Pain/Maim cannot run to local physical drives	**Cons:** • Catapult cannot see mapped network shares • GUI cannot see mapped network shares	**Cons:** • UAC is off • Windows 8 Metro Live Tiles don't work
Work-arounds: • For GUI: right-click on GUI -> choose "Run as administrator..." • For CLI: "Run as administrator..." a cmd.exe -> run the tools from within that console		
NOTE1: UAC OFF has been the normal (and one-and-only supported) mode of operation before MLTT 6.0.1. **NOTE2:** This is an issue only on local systems. If MLTT running on a Windows system is accessed only remotely using the GUI or Catapult on another system, then whether or not that remote system has UAC on/off is not an issue.		

Testing Concepts

This section discusses some of the system and network planning considerations that you must account for to use MLTT effectively.

Target Considerations

You need to consider the capabilities and characteristics of a target device when setting up a test. Queue depth, block size, and I/O modes are the most influential test parameters from the target perspective.

The most prominent target limitation, particularly when testing hard drives, is queue depth. Excessive queue settings that are specified with the intent of providing maximum stress may result in minimal throughput due to queue full conditions on the target.

You also need to consider a target's caching abilities when setting up a test, especially on RAID controllers with large amounts of cache memory. You might want to keep I/O characteristics such that the target is able to service requests within cache, for the purpose of performance or stress testing of a host system or an interconnecting device such as a switch. On the other hand, when testing the target itself, you should include tests that overrun cache boundaries and force frequent commits to the hard drives. This can be accomplished with combinations of queue (thread), block, and file size parameters.

There are a variety of I/O modes in MLTT, including static queuing, random access, and full stroke target coverage. Comprehensive testing of target systems should include exposure to these modes.

Important: To insure that the maximum possible throughput to a target is realized, you should run MLTT to physical devices (raw access) whenever possible. Running I/O traffic to a file system involves several layers of overhead at the host system, which results in a lower stress load on the target.

Windows platforms provide raw access through physical drive links (for example: `\\.\physicaldrive1`).

MLTT uses O_DIRECT by default on Linux systems with kernel version 2.6 or higher for non-cached I/O. However, on earlier Linux kernels, it is necessary to bind the block devices to a "raw" device to achieve non-cached I/O. See "man raw" on a Linux kernel 2.4.x system for more details.

Example:

```
raw /dev/raw/raw1/dev/sdb
```

When you use Catapult to start physical I/O tests on a Linux kernel 2.4 system, this binding is made for you automatically.

Raw access on Solaris platforms is performed through the "rdsk" device path to a drive slice (for example: `/dev/rdsk/c1t0d0s2`).

Protocol Analyzers

While Medusa Labs Test Tools (MLTT) are designed to report conditions as accurately as possible from the application level, we cannot overemphasize the importance of using in-line protocol analyzers or traffic monitors whenever possible. In our experience, a substantial number of defects or deficiencies are overlooked in product development due to anomalies that are not readily visible at the application level. An analyzer is essential to detecting underlying items of interest, such as I/O fragmentation and recoverable errors, and verifying performance numbers.

A powerful feature of MLTT is the ability to send an I/O with a special value for analyzer triggering. Fault conditions which would be difficult, if not impossible, to debug and to find the root-cause from the application level can easily be captured in a trace and analyzed in detail.

TraceView Support

Some JDSU traffic generator products add some records into the data portion of the traffic. This is the case for the JDSU Load Tester and Medusa Labs Test Tools. MLTT adds a special record at every 512-byte boundary of the SCSI data.

The following choices are available in JDSU Analyzer TraceView:

Don't Decode JDSU Signatures

This option disables the decoding of the JDSU special records. This is the default setting.

Medusa Labs Test Tools I/O Signature

This option enables the decoding of the MLTT Signature records every 512 bytes in the SCSI data. This switch enables decoding of these special records.

Medusa Labs Test Tools I/O Signature/Timestamp in seconds

This option is the same as the previous one, except that an additional 32-bit timestamp is decoded at the end of the record. This timestamp is added to the record if specified by command-line arguments within MLTT when the capture is created. Refer to “-U I/O Signature Timestamp Units” on page 142 for information on specifying the timestamp addition to the I/O signature in seconds using the `-U` command.

Medusa Labs Test Tools I/O Signature/Timestamp in milliseconds

This option is the same as the previous one, except that an additional 16-bit millisecond resolution timestamp is added after the 32-bit timestamp. This timestamp is also added by command-line argument. Refer to “-U I/O Signature Timestamp Units” on page 142 for information on specifying the timestamp addition to the I/O signature in milliseconds using the `-Um` command.

Chapter 2

Using the Graphical User Interface

In this chapter:

Using the Medusa Labs Test Tools GUI

The Medusa Labs Test Tools (MLTT) Graphical User Interface (GUI) provides a quick, visual system to run MLTT. The basic steps for running a test with the GUI are:

- Selecting the targets.
- Selecting a configuration or configuring a new set of test parameters.
- Running the test.
- Viewing the output results.

Launching the Medusa Labs Test Tools

When MLTT was installed, it is likely that the Medusa Labs Test Tools shortcut icon was installed on the desktop. Clicking this icon is the quickest way to launch MLTT. However, you can also launch the MLTT application by:

1 Clicking the Windows Start menu.

2 Choosing **Programs** > **Medusa Labs Test Tools** > **Medusa Labs Test Tools**.

Note: For Windows 8, select the **Start** button and the select the **Medusa Labs Test Tools** icon.

The Medusa Labs Test Tools main window opens.

Setting Up a Performance Test

Setting up a simple performance test includes selecting the target (or targets), selecting/creating and editing the configuration, running the test, and viewing the test output.

Selecting the Target

You can select targets from a list of targets that are available for testing with MLTT.

1 After creating a test plan, select the targets from the list displayed in the **Targets** area.

You can expand/collapse the drive categories by clicking the plus/minus sign on the left of each category.

2 To select the targets, click the target and drag it to the test plan you created in the **Test Plans** area.

Note: You can also select the targets first and drag/drop the target into the **Test Plan Browser** (see page 37) to automatically create a Test Plan.

Some grayed-out targets cannot be selected because they include an OS or active partition. This protection mechanism keeps critical data from being overwritten during testing.

Caution: Physical access is destructive to the data on the target and can overwrite the partition data.

You can select a group of drives, such as **File System,** to select all the drives of that category.

To view the information for a target, right-click the target and select **Properties**.

Selecting or Creating the Configuration

You can select either an existing configuration or create a new configuration for testing with MLTT.

1. Select an existing configuration in the folder of **Medusa Sample Configurations**, or create a new configuration in the **Configurations** area of the **Medusa Labs Test Tools** main window.
2. On the **Configurations** area, select the folder where you want the new configuration to be located. For example, select the User Configurations folder to locate your new configuration in it.

   The **Medusa Sample Configurations** folder is read-only. When creating a new configuration, select the **User Configurations** folder or create a new folder by clicking the **New Folder** button .
3. Click the black arrow at the right of the **New Configuration** button to open the drop down menu and then select the desired configuration type.

   The new configuration is listed in the folder. You can right-click the new configuration and click **Rename**, or click and pause on the name to rename it.

   You can also create a new configuration from an existing configuration. To copy a configuration from the Medusa Sample Configurations, right-click the configuration and choose **Copy** from the pop-up menu. Then select the new folder, right-click, and choose **Paste**. This method is particularly helpful when you only want to edit a few settings of an existing configuration to create a new configuration.
4. Double-click the new configuration to edit the options. The **Configuration Editor** window is displayed.

   For information about configuration editor settings, see Chapter 3, “Using the Configuration Editors”.
5. Click **OK**.
6. To select a configuration, click the desired configuration and drag it to the test plan on the **Test Plans** area.

Running the Test

Run the test based on the targets and configuration you selected.

1 Click the **Start** button. Make sure that you have selected the test plan that you want to run.

The test begins on the selected targets for the selected configurations and runs until the specified duration expires, or the test is stopped manually.

Clicking the **Next** button stops the configuration that you are currently running and will run the next batch of configurations.

2 Click the **Stop** button to manually stop all configurations.

Viewing the Test Output, Exporting Test Summaries, and Deleting Test Plan History

Test results are displayed in the **Test Analysis** tab as each test configuration has completed testing. For test configurations with multiple iterations or test plans with multiple configurations, test results are displayed in **Test Analysis** tab as each configuration completes.

1 To see the logged results, check either the **History Summaries** or the **History Tests** panes.

2 To export test summaries:

- In the **History Summaries** pane, select the test plan and in the **File** menu, select the **Export Selected Histories...** to open a dialog box that allows you to select where the test summaries are to be saved. The history file is saved with a ".his" extension. This file can be moved to another system where it can be viewed using MLTT.

Note: The saved .his file can be viewed and analyzed using another system running MLTT. To view this file, move the .his file to the other system, from the **File** menu, select **Import Histories...**, and use the **Select Histories** dialog box to locate and select the .his file which will import the file.

- In the **History Summaries** pane, right-click on the test plan and select **Export selected summaries to CSV...** to open a dialog box that allows you to select where the test summaries are to be saved.
- In the **History Tests** pane, right-click on a test and select **Export all summaries to CSV...** or select the summaries that you want to save, right-click and select **Export selected summaries to CSV...**.

  Each selection opens a dialog box that allows you to select where the test summaries are to be saved.

3 To delete a test plan history, in the **History Summaries** pane, select the test plan and press the **Delete** key or right-click and select **Delete Selected History**.

Medusa Labs Test Tools GUI

The Graphical User Interface (GUI) implements options available from the command line version of the application.

The GUI window is made up of:

- “Medusa Labs Test Tools Menus” on page 25
- “Test Planning Tab” on page 28
- “Test Running Tab” on page 45
- “Test Analysis Tab” on page 50

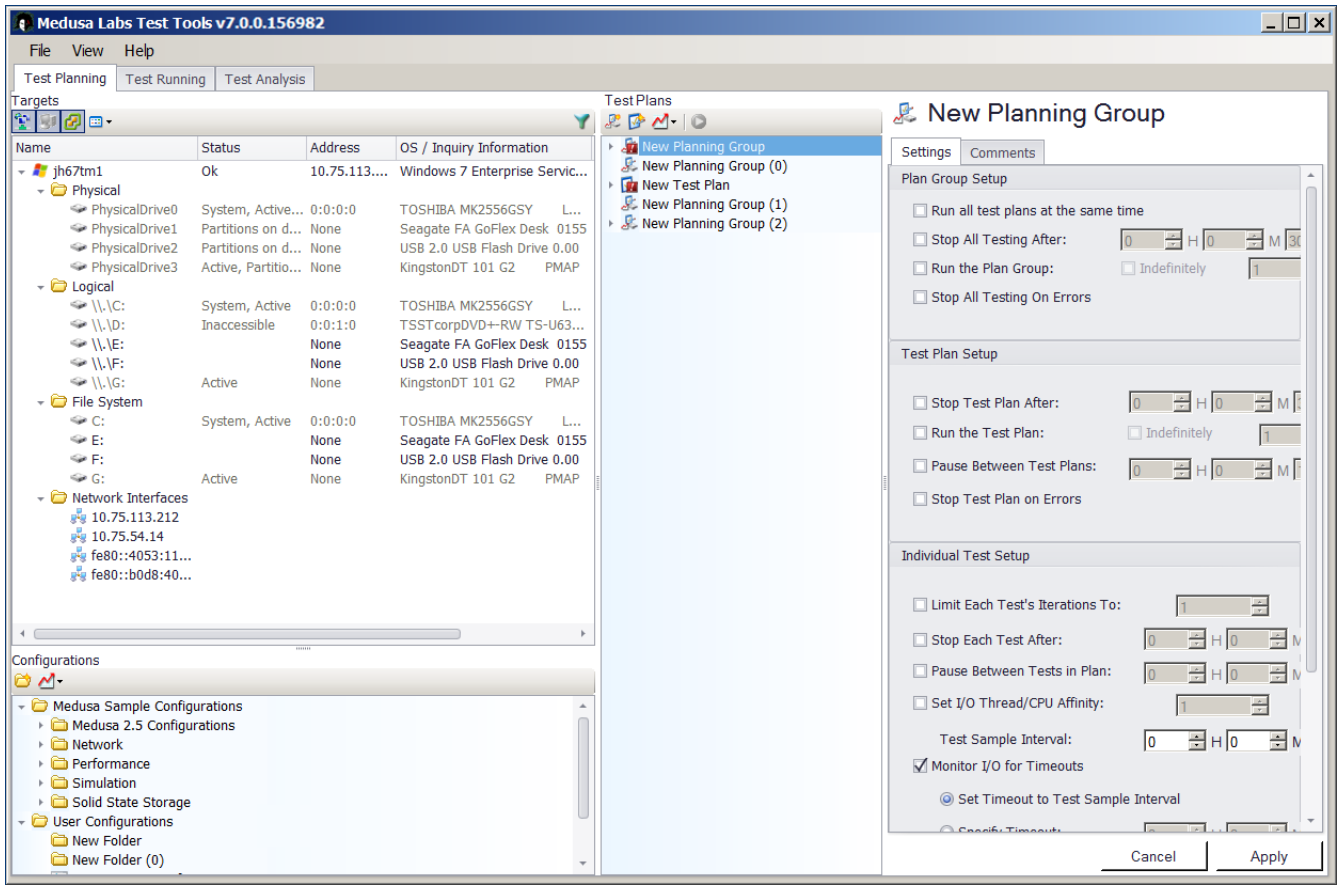
Refer to “GUI Overview” for a brief overview of the GUI.

For details about the command line equivalents of the GUI options, see Chapter 4, “Using the Command Line Switches’’.

GUI Overview

The Medusa Labs Test Tools (MLTT) main window is the starting point for using the GUI.

Figure 4: Medusa Labs Test Tools GUI Window



The Medusa Labs Test Tools Main window contains the following components (refer to Figure 4):

Menu Bar (See page 25)

The **Menu** bar contains the **File**, **View**, and **Help** menu items. For more information about the MLTT menus, see "Medusa Labs Test Tools Menus".

Test Planning Tab (See page 28)

The Test Planning tab contains the following panes:

- The **Targets** pane contains four buttons (**Show/Hide Remote Systems**, **Show/Hide Offline Systems**, **Show/Hide VMware ESX(i) Servers**, and **Change Visibility of Targets**) and the **Target Categories** section. For more information on target selection, see "Targets Area".
- The **Configurations** pane contains the **New Folder** button, the **New Configuration** button, and the **Configurations** section. For more information on configuration settings, see "Configurations Area".
- The **Test Plans** pane contains the following parts:
  - Test Plan buttons: **New Planning Group, New Test Plan**, **New Configuration**, and the **Select a test to start/Press start to begin tests** button
  - **Test Plan/Planning Group Browser** where you create, edit, and delete test plans and planning groups
  - **Test Plan/Planning Group/Configuration Properties** pane where you can edit or customize the properties Test Plans, Planning Groups, or Configurations.

Test Running Tab (See page 45)

The Test Running tab contains the following sections:

- Test control buttons: **Stop all testing**, **Stop currently selected test**, and **Move to the next test**
- **Running Test list** lists the running test plans
- **Console View** shows the results of the selected test plan or its components
- **Speedometers** displays the real-time speed of the tests

Test Analysis Tab (See page 50)

The Test Analysis tab contains the following sections:

- **History Summaries**
- **History Tests**
- **History Summaries (or Tests) Information**

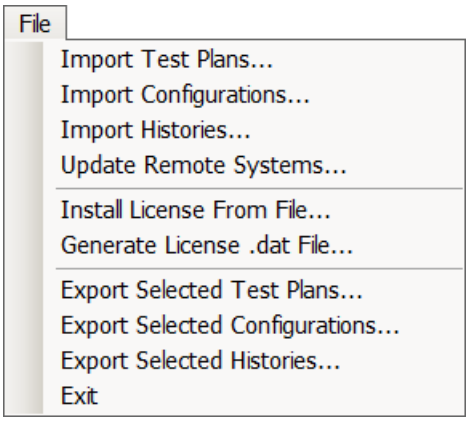
Medusa Labs Test Tools Menus

The three menus available with the MLTT GUI are **File**, **View**, and **Help**.

File Menu

The **File** menu (Figure 5) lets you import test plans, configuration, and history files, install license from file, update remote systems, export selected test plans and selected configurations, and close MLTT.

Figure 5: File Menu



Import Test Plans...

This option opens the **Select Test Plans** dialog box where you can browse and select a previously saved test plan to import. You can import Medusa Lab Test Plans (.sdf files), Legacy Test Plans (.mltp files), and IOMeter Configuration Files (.icf and .txt files). Note that IOMeter configuration files are replicated as closely as possible and targets are not imported.

Import Configurations...

This option opens the **Select Configurations** dialog box where you can browse and select a previously saved configuration to import.

Import Histories...

This option opens the **Select Histories** dialog box where you can browse and select a previously saved history to import. This file has a .his file extension. This feature allows you to export data (using the **File** menu's **Export Selected Histories...** selection) from the system that has run the test and view it on another system.

Update Remote Systems...

This option opens the **Install Updates** dialog box where you can install Tools updates to remote systems.

Before updating any remote system, please ensure that all MLTT related files are closed before using this feature.

Install License From File...

This option opens the **Select License File** dialog box where you can browse and select a license file (.lic file) to install. This is equivalent to running the command *pain -Z#*.

Generate License .dat File...

This option opens a browser window where you can browse and select a location where the generated license file will be saved.

Export Selected Test Plans...

This option allows you to export a selected test plan from the **Test Planning** area. This option opens the **Export Test Plans** dialog box where you can select the folder to export the selected test plan.

Export Selected Configurations...

This option allows you to export a selected test configuration from the **Configurations** area. This option opens the **Export Configurations** dialog box where you can select the folder to export the configuration.

Export Selected Histories...

This option allows you to export a selected test summary from the **History Summaries** area of the Test Analysis tab. This option opens the **Export Histories** dialog box where you can select the folder to export the history summary. The history file is saved with a ".his" extension. This file can be moved to another system where it can be imported using the MLTT **File** menu's **Import Histories...** selection. The file can then be viewed on the system.

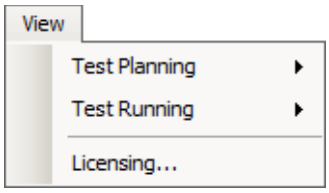
Exit

Exits Medusa Labs Test Tools.

View Menu

The **View** menu (refer to Figure 6) gives you the option to show or hide sections of the GUI main window. Options for the View menu are:

Figure 6: View Menu



Test Planning

This allows you to show or hide remote systems. Select **Show Remote Systems** to have the remote systems available in the **Targets** pane of the **Test Planning** tab.

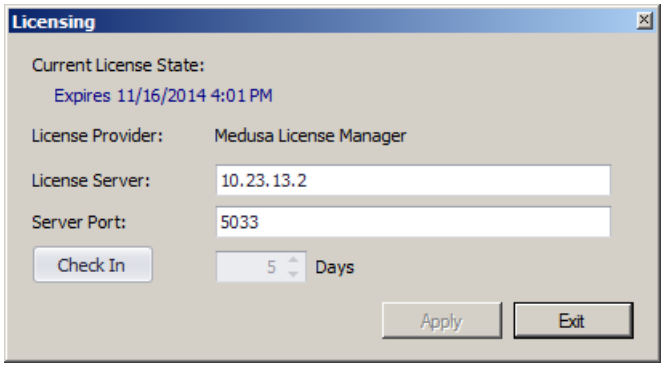
Testing Running

This allows you to show or hide the **Console View** and the **Speedometers** in the **Test Running** tab.

Licensing...

This allows you to show/hide the **Licensing** dialog where you can see the status of the license, check in or check out a license and see additional information about the current license status.

Figure 7: Licensing Dialog



Note: The Licensing dialog box for any system may be accessed. Refer to "Accessing System Licensing Information" on page 31 for instructions.

Help Menu

This menu provides a link to the user's guide and the MLTT software information.

Figure 8: Help Menu

User's Guide

This selection displays the *Medusa Labs Test Tools Suite User's Guide*.

About

This selection displays information about MLTT.

Test Planning Tab

The Test Planning tab has three main areas:

Targets area lists the targets available for testing with MLTT starting on page 28.

Configurations area is used to set up or select the configuration to use for testing starting on page 32.

Test Plans area is used to set up and select the test plan starting on page 34.

Targets Area

The **Targets** area lists the targets available for testing with MLTT.

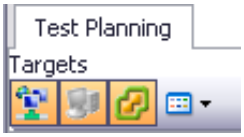
The target can be a file, logical drive, or physical drive that resides in the host system or is externally attached via SCSI, USB, FireWire, LAN, SAN, Sockets, and others.

The **Targets** area contains the following:

Targets View Buttons

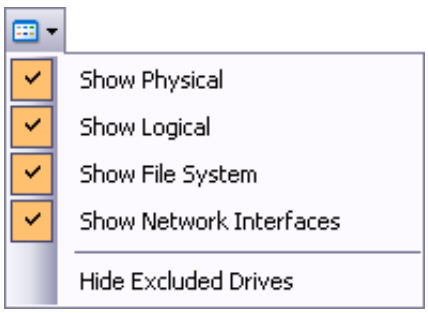
These buttons allow you to choose the type of targets to show in the **Targets Categories** section.

Figure 9: Targets View Buttons



.

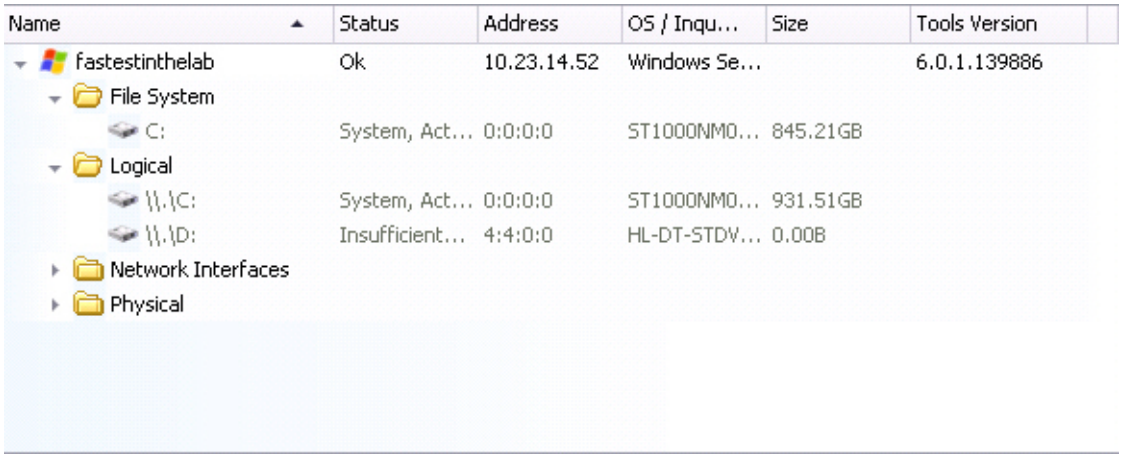
Table 3: Targets View Button Descriptions

	This toggles to allow you to show or hide remote systems in the **Target Categories** section.
	This toggles to allow you to show or hide offline systems in the **Target Categories** section.
	This toggles to allow you to show or hide VMware ESX(i) servers in the **Target Categories** section.
	From the drop-down menu, you can show or hide physical drives, logical drives, and file systems that reside in the host system displayed in the **Target Categories** section. Targets can also be remote systems or systems that are externally attached via SCSI, USB, FireWire, LAN, SAN, and others.

Target Categories Section

The **Target Categories** section lists the targets that are available for testing with MLTT. The details for each of the target are shown in columns.

Figure 10: Target Categories Section



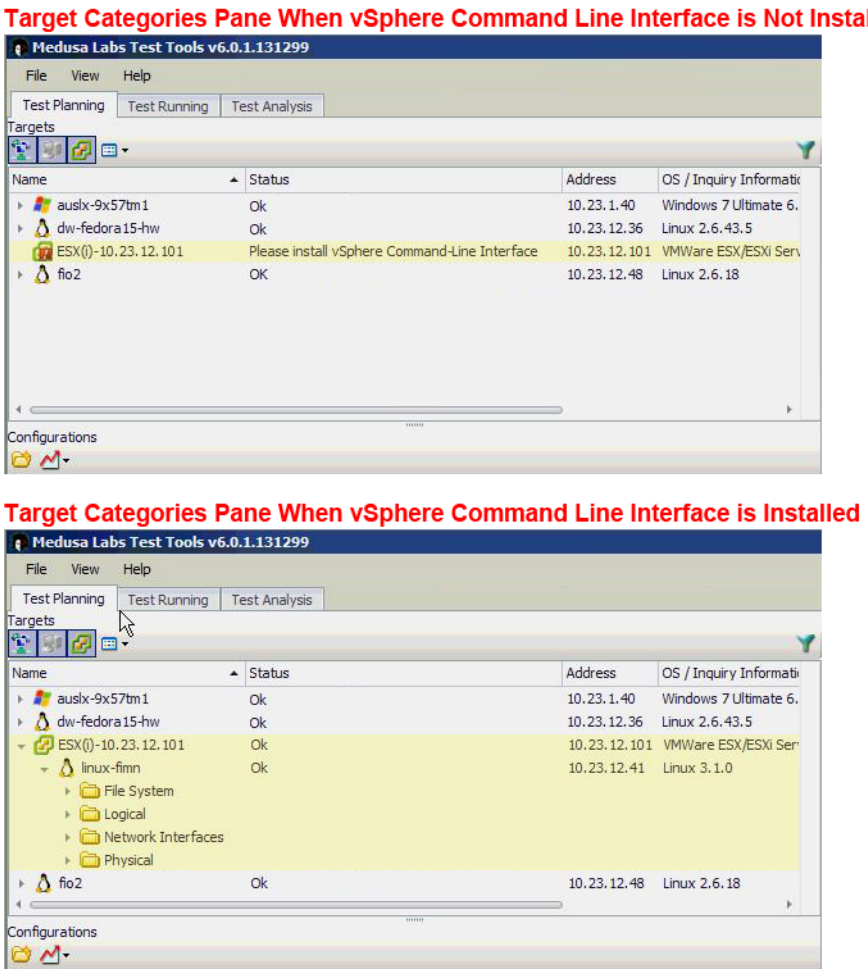
To show or hide the target objects such as File System, Logical, Network Interfaces, or Physical, click the arrow icons (▼ ▶) at the left edge of the target.

- When the icon is an arrow pointing right (▶), the target objects are hidden; click the arrow to show the objects.
- When the icon is an arrow pointing down (▼), the target objects are shown; click the arrow to hide the objects.

Some grayed-out targets cannot be selected because they have a target exclusion. These exclusions correspond with the ones used by catapult. This protection mechanism keeps critical data from being overwritten during testing. To override device exclusions for these targets, right-click on the target or IP address (either IPv4 or IPv6 addresses) and then click **Override device exclusions**. The system exclusion cannot be overridden.

The ESX-server-to-VM relationship can be displayed in the Target Categories pane. However, you must first obtain and install the VMware vSphere Command Line Interface (vSphere CLI) tools that are available from the support and download page on the VMware website (http://www.vmware.com/). Figure 11 shows the ESX displayed in the Target categories pane. The upper illustration shows the ESX when the vSphere CLI is not installed while the lower illustration shows the ESX after the vSphere CLI has been installed.

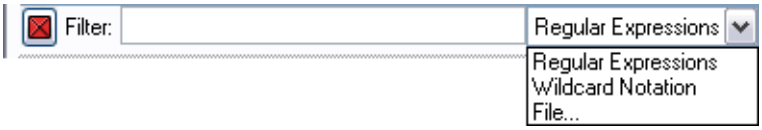
Figure 11: vSphere Command Line Interface



To filter systems according to host name or IP address:

1 Right-click on the **Target Categories** section and click **Filter Hosts**. The **Filter** search box (Figure 12) is displayed below the **Target Categories** section.

Figure 12: Filtering Hosts



2 Type the host name or IP address (either IPv4 or IPv6 addresses) to show only the host or hosts according to the filter you typed.

3 Select the type of filter, whether a regular expression, a wildcard notation, or a file, from the drop-down list. If you select File, you need to browse for a plain text file with one host name or IP address per line.

4 Click to close the **Filter** textbox.

To connect to a remote subnet, right-click on the **Target Categories** section and click **Connect to Remote Subnet** and enter the IP address of a system on the remote network that is running MLTT.

MLTT will not automatically connect to systems outside the local subnet. If you want to connect to another subnet, they must install the tools on a system there and use either catapult or the GUI to connect to that system.

To display the device characteristics of a target, right-click on a target in the **Target Categories** section and click **Properties**. The **Properties** window for that selected device will appear.

For hosts, you can also click on the **Configure** button to launch the **Licensing** window. The **Configure** button in the **Properties** window is only available for hosts.

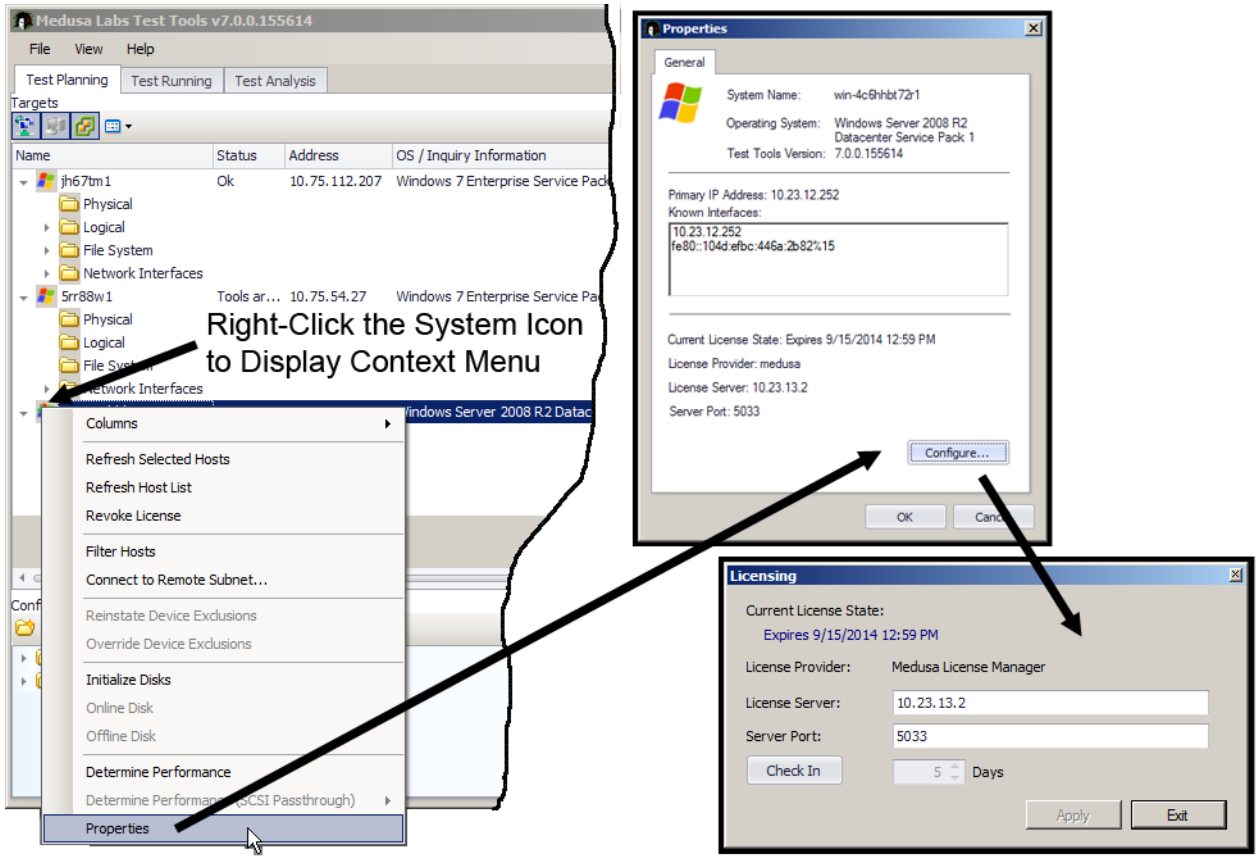
On versions of Windows 2003 R2 and later, devices can be brought online or offline. When devices are offline, they cannot be written to or read from and so are not good for testing. The GUI can bring drives online so that they can be tested.

To bring a device online or offline, right-click on a target in the **Target Categories** section and click **Online Disk**, **Offline Disk**, or **Initialize Disks**.

Accessing System Licensing Information

The licensing information on the **Licensing** dialog box can be accessed for any system by right-clicking the system icon, selecting **Properties**, selecting the **Configure...** button.

Figure 13: Accessing System Licensing Information



Configurations Area

The **Configurations** area is used to create and manage new configurations. Both new and existing configurations may also be edited from this area. These edits are performed using the configuration editors. These editors are described in detail in Chapter 3, “Using the Configuration Editors’’ starting on page 61.

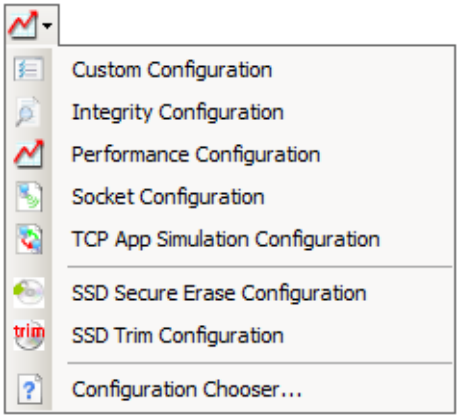
New Folder Button

This button allows you to create a folder where you can place your new configurations.

New Configuration Button

This button opens the following drop-down list of configurations. Selecting one of these configurations creates a new blank configuration in the **User Configurations** folder that is located in the area below the buttons. You can also select the **Configuration Chooser** from the menu to open the **Configuration Chooser** window.

Figure 14: Create New Configuration Button



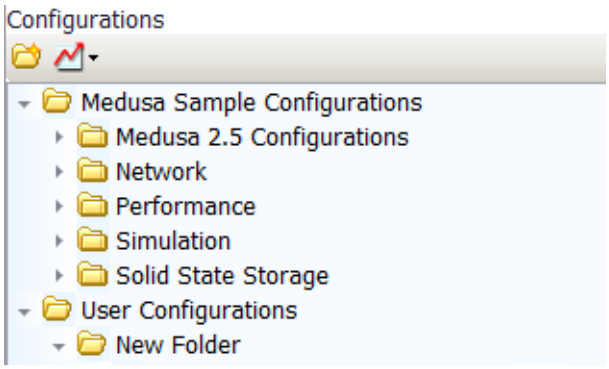
Note: The configuration editors are described in detail in Chapter 3, “Using the Configuration Editors’’ starting on page 61.

You can also access the list of configurations by right-clicking in the Configurations pane and selecting **New Configuration**. Refer to Figure 21 on page 36 for a menu that is similar.

Configurations section

The **Configurations** section lists the folders that contain the sample configurations and configurations you have created. The sample configurations may not be edited.

Figure 15: Configurations Section



By default there are two main Configurations folders listed in the Configurations section: the **Medusa Sample Configurations** and the **User Configurations**. You can add more sub-folders to the **User Configurations** folder by clicking the **Create New Folder** button.

The **Medusa Sample Configurations** cannot be edited. However, you can copy a sample configuration, paste it in the **User Configurations** folder, and then edit the pasted copy.

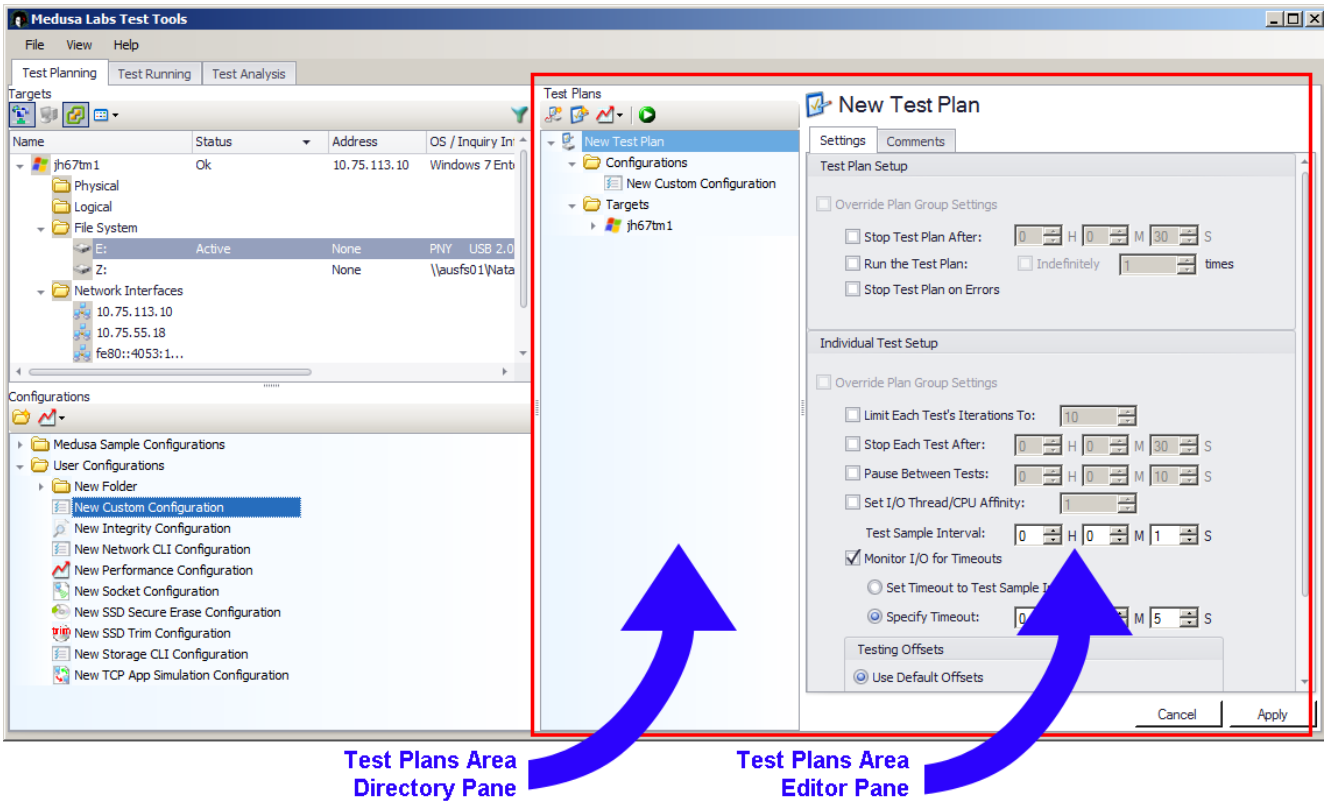
To rename any of the folders or configurations listed in the **User Configurations** folder, select the folder or configuration and click it again to edit the name.

To edit a configuration listed in the **User Configurations** folder, double-click the icon of the configuration and the configuration editor will open.

Test Plans Area

The Test Plans area occupies the right portion of the MLTT application window. This area has a directory pane (see “Test Plans Directory Pane” on page 34) and an editor pane when an object (planning group, test plan, or configuration) is selected in the Test Plans directory pane (see “Test Plans Editor Pane” on page 39).

Figure 16: Test Plans Area



Test Plans Directory Pane

The Test Plans directory pane consists of the four buttons near the top of the pane (shown in Figure 17) and the Test Plan Browser which displays all the available planning group and test plan objects.

Figure 17: Directory Pane Buttons

From left to right, these buttons are the **New Planning Group** button, the **New Test Plan** button, the **New Configuration** button, and the **Press start to begin tests** button.

New Planning Group Button

This button allows you to create a new planning group that can be used to group test plans which allows you to run different configuration and target pairings. When the button is selected, the new planning group is listed in the directory pane and the new planning group editor is displayed in the editor pane.

Figure 18: New Planning Group Button

New Test Plan Button

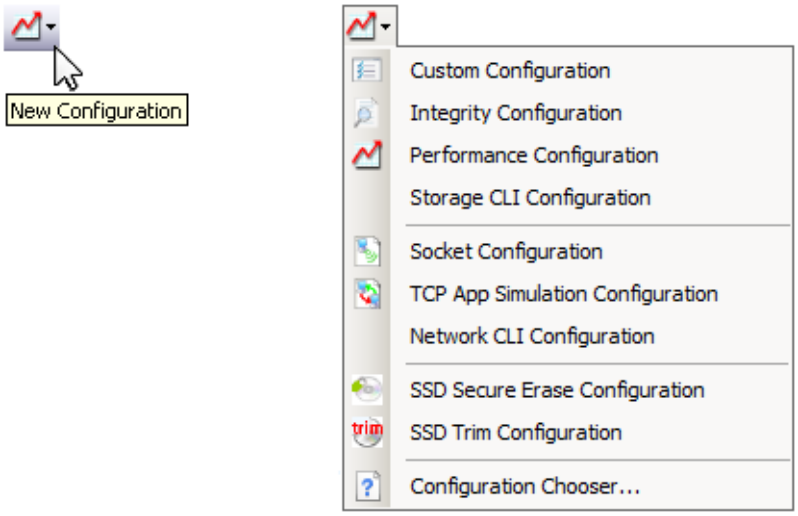
This button allows you to create a new test plan. When the button is selected, the new test plan is listed in the directory pane and the new test plan editor is displayed in the editor pane.

Figure 19: New Test Plan Button

New Configuration Button

This button opens a dropdown list of configurations selections, such as custom, integrity, performance, socket, and TCP App Simulation. There are also two command line (Storage CLI and Network CLI), the SSD Secure Erase, and the SSD Trim configurations available.

Figure 20: New Configuration Button

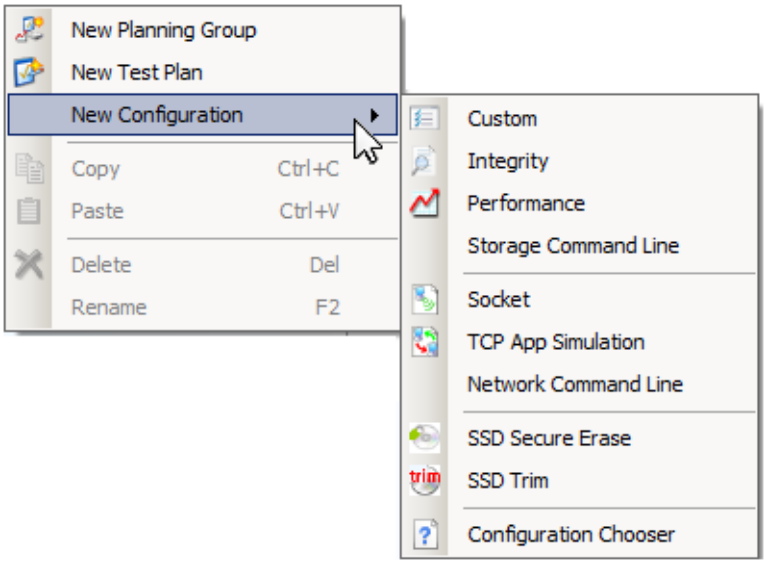


Selecting a configuration from this list allows you to create a new configuration. When a configuration is selected, the new configuration is listed in the directory pane and the new configuration editor is displayed in the editor pane. Detailed descriptions for each of these configuration editors are available in Chapter 3, “Using the Configuration Editors” starting on page 61.

You can also select the **Configuration Chooser...** from the dropdown list to open the **Configuration Chooser** window. When the configuration is chosen, the new configuration is added to the selected test plan in the directory pane and displayed in the editor pane.

You can also access the list of configurations by right-clicking in the Test Plans pane and selecting **New Configuration**.

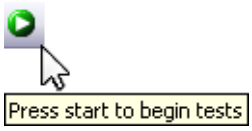
Figure 21: Right Click Test Plans Pane



Press start to begin tests Button

This button starts the testing of the selected group or test plan object in the directory pane.

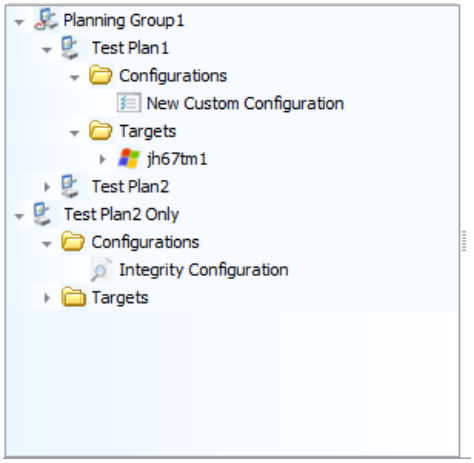
Figure 22: Press start to begin tests Button



Test Plan Browser

The **Test Plan Browser** (shown in Figure 23) displays all the planning groups, test plans, configurations, and targets that you created or copied to the directory pane.

Figure 23: Test Plan Browser



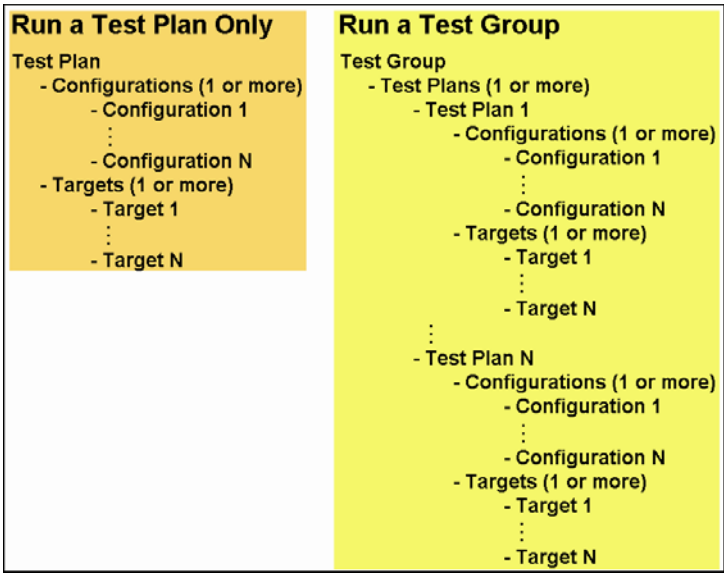
To show or hide the test plan browser objects such as test plan group, test plan, configuration, or target, click the arrow icons (▾ ▸) at the left edge of the target.

- When the icon is an arrow pointing right (▸), the objects are hidden; click the arrow to show the objects.
- When the icon is an arrow pointing down (▾), the objects are shown; click the arrow to hide the objects.

Note: Icons can be moved by clicking and dragging the icon to another location in the Test Plan Browser. For instance, you may select a configuration in one test plan and move it to another test plan. The location and order of the icons are maintained after MLTT is closed and reopened.

Each of the objects (planning groups, test plans, configurations, and targets) can be viewed as a container for performing a test and each has a hierarchy related to the others. See Figure 24.

Figure 24: Test Plan Browser Hierarchy



The Planning Group is the highest level in the hierarchy and contains one or more test plan. A planning group allows you to run multiple test plans. However, it is not required if you want to run only one test plan. You can edit the Planning Group in the Test Plans Area Editor Pane. Refer to “Planning Group Editor” on page 40.

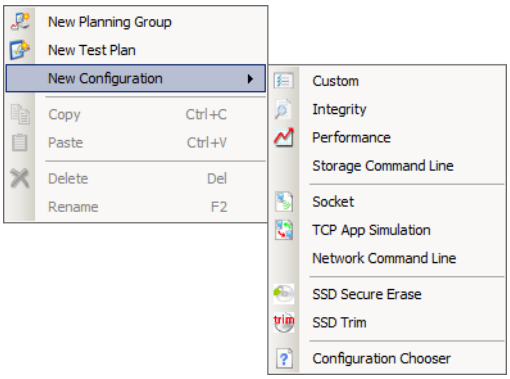
The Test Plan contains the configurations (1 or more) and the targets (1 or more). When the Test Plan is run, it will run all of its Configurations against all of its Targets. A Test Group containing the Test Plan is not required to run the one Test Plan only. You can edit the Test Plan in the Test Plans Area Editor Pane. Refer to “Test Plan Editor” on page 42.

At least one Configuration is required for a Test Plan. You can edit the Configuration in the Test Plans Area Editor Pane. Refer to “Configuration Editor” on page 44.

At least one Target is required for a Test Plan. Targets are dragged to the Test Plan from the Targets area. Refer to “Target Categories Section” on page 29. When selected, the target information is displayed in the Test Plans Area Editor Pane.

New planning groups, test plans, and configurations can also be added by right-clicking in the test plan browser to display the context menu shown in Figure 25.

Figure 25: Test Plan Browser Context Menu



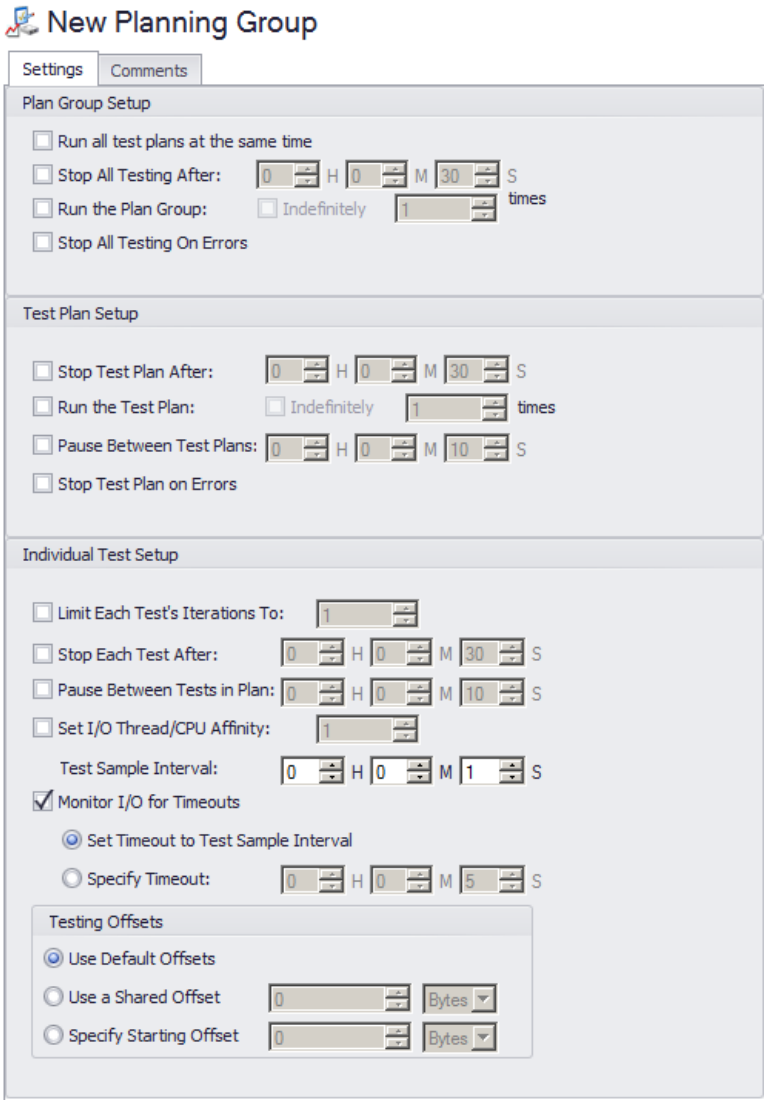
Test Plans Editor Pane

The Test Plans editor pane allows you to set up the properties of the object (planning group, test plan, or configuration) that is selected in the Test Plans directory pane. For example, if a test plan is selected in the directory pane, the properties for that test plan are displayed in the editor pane so that you may edit the properties if desired. For details on each editor, refer to “Planning Group Editor” located below, “Test Plan Editor” on page 42, or “Configuration Editor” on page 44.

Planning Group Editor

The **Planning Group** editor is displayed in the Test Plans editor pane when a planning group is selected in the directory pane (or when the **New Planning Group** button is selected.) The **Planning Group** editor allows you to edit the following planning group properties:

Figure 26: Planning Group Editor



Settings Tab

The **Settings** tab displays the properties for the selected planning group. The planning group properties are:

Plan Group Setup – These options specify the test duration and the number of iterations.

Run all test plans at the same time – Select this check box

Stop All Testing After – Select this check box and edit the number or click the up and down arrows to set the time to limit the run time of the test plans in the selected group.

Run the Plan Group – Select this check box and edit the number or click the up and down arrows to set the number of times the test group containing the test plans will be repeated.

Stop All Testing On Errors – Select this option to stop the tests if errors are detected.

Test Plan Setup – These options specify the test duration and the number of iterations.

Stop Test Plan After – Select this check box and edit the number or click the up and down arrows to the time to limit the run time of the selected test plan.

Run the Test Plan – Select this check box and edit the number or click the up and down arrows to set the number of times the test plan will be repeated.

Pause Between Test Plans - Select this check box and edit the number or click the up and down arrows to the time to pause the run after a test plan has run before starting the next test plan.

Stop Test Plan on Errors - Select this option to stop the test if errors are detected.

Individual Test Setup – These options configure the behavior for each of the test plans.

Limit Each Test's Iterations To – An iteration, called a file operation (FOP), is a complete write and read of an entire file or specified extent on a logical or physical drive. If this option is not selected, the test will run until manually stopped, or a critical error is encountered.

Stop Each Test After – Select this option to set the duration of each test in the selected test plan.

Pause Between Tests in a Plan – Select this option to set how long the pause will be between each test in the test plan.

Set I/O Thread/CPU Affinity – Select this option to specify the number of CPUs to use for I/O threads. In addition to limiting the number of CPUs used, this option causes a thread to always run on the same CPU. The number of CPUs specified must be equal to or less than the number of CPUs on the system and equal to or less than the total number of I/O threads.

Test Sample Interval – Select this option to set the time between performance samples. Performance samples show the continuing test performance and are written to the log file. Edit the number or click the up and down arrows to specify the performance sample interval.

Monitor I/O for Timeouts – Select this option to enable the I/O monitoring mode. A warning will be displayed when I/Os are not completed before the specified number of seconds. By default, warnings will appear when a completion exceeds the performance sample time (5 seconds is the default sample time.) The I/O monitoring feature will report both complete I/O halts and individual stuck I/Os.

This mode can also be used to catch I/O disruptions on an analyzer. If this mode is used with the option to continue testing and generate a trigger, an I/O trigger is sent when a halt or stuck I/O is detected.

Set Timeout to Test Sample Interval – Select this option to set the timeout to the sample interval specified in the **Test Sample Interval** field.

Specify Timeout – Edit the number or click the up and down arrows to specify a timeout or clear the check box to disable monitoring.

Testing Offsets - These options let you specify the testing offsets.

Use Default Offsets - uses the default offset.

Use a Shared Offset - allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently.

Specify Starting Offset - specifies the starting offset number. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target device.

Comments Tab

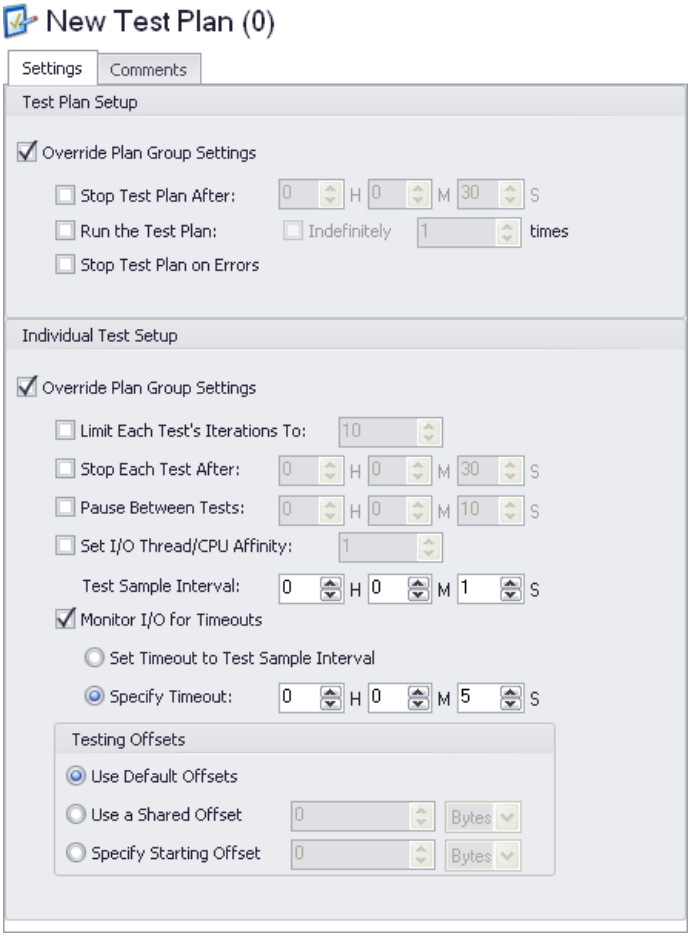
Enter your comments for the configuration in the comment box of the **Comments** tab

Apply button – Click this button when you are done editing the planning group properties.

Test Plan Editor

The **Test Plan** editor is displayed in the Test Plans editor pane when a test plan is selected in the directory pane (or when the **New Test Plan** button is selected.) The **Test Plan** editor allows you to edit the following test plan properties:

Figure 27: Test Plan Editor



Settings Tab

The **Settings** tab displays the properties for the selected test plan. The test plan properties are:

Test Plan Setup – These options specify the test duration and the number of iterations.

Override Plan Group Settings – Select this check box to override the Test Plan Setup settings if they were setup in the Planning Group using the settings now displayed.

Stop Test Plan After – Select this check box and edit the number or click the up and down arrows to the time to limit the run time of the selected test plan.

Run the Test Plan – Select this check box and edit the number or click the up and down arrows to set the number of times the test plan will be repeated.

Stop Test Plan on Errors - Select this option to stop the test if errors are detected.

Individual Test Setup – These options configure how the each of the test plans behavior.

Override Plan Group Settings – Select this check box to override the Individual Test Setup settings if they were setup in the Planning Group using the settings now displayed.

Limit Each Test's Iterations To – An iteration, called a file operation (FOP), is a complete write and read of an entire file or specified extent on a logical or physical drive. If this option is not selected, the test will run until manually stopped, or a critical error is encountered.

Stop Each Test After – Select this option to set the duration of each test in the selected test plan.

Pause Between Tests – Select this option to set how long the pause will be between each test in the test plan.

Set I/O Thread/CPU Affinity: – Select this option to specify the number of CPUs to use for I/O threads. In addition to limiting the number of CPUs used, this option causes a thread to always run on the same CPU. The number of CPUs specified must be equal to or less than the number of CPUs on the system and equal to or less than the total number of I/O threads.

Test Sample Interval – Select this option to set the time between performance samples. Performance samples show the continuing test performance and are written to the log file. Edit the number or click the up and down arrows to specify the performance sample interval.

Monitor I/O for Timeouts – Select this option to enable the I/O monitoring mode. A warning will be displayed when I/Os are not completed before the specified number of seconds. By default, warnings will appear when a completion exceeds the performance sample time (5 seconds is the default sample time.) The I/O monitoring feature will report both complete I/O halts and individual stuck I/Os.

This mode can also be used to catch I/O disruptions on an analyzer. If this mode is used with the option to continue testing and generate a trigger, an I/O trigger is sent when a halt or stuck I/O is detected.

Set Timeout to Test Sample Interval – Select this option to set the timeout to the sample interval specified in the **Test Sample Interval** field.

Specify Timeout – Edit the number or click the up and down arrows to specify a timeout or clear the check box to disable monitoring.

Testing Offsets - These options let you specify the testing offsets.

Use Default Offsets - uses the default offset.

Use a Shared Offset - allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently.

Specify Starting Offset - specifies the starting offset number. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target device.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab

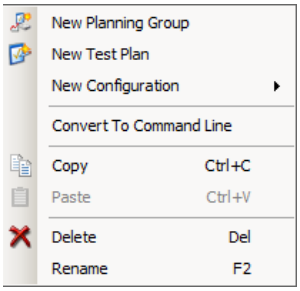
Apply button – Click this button when you are done editing the test plan properties.

Configuration Editor

The **Configuration** editor is displayed in the Test Plans editor pane when a configuration is selected in the directory pane (or when the **New Configuration** button is selected.) The **Configuration** editor allows you to edit the same configuration properties that you can edit in the configuration editor.

In the **Test Plans** browser pane, when a configuration is right-clicked, the context menu displays a **Convert to Command Line** option.

Figure 28: Convert to Command Line Option



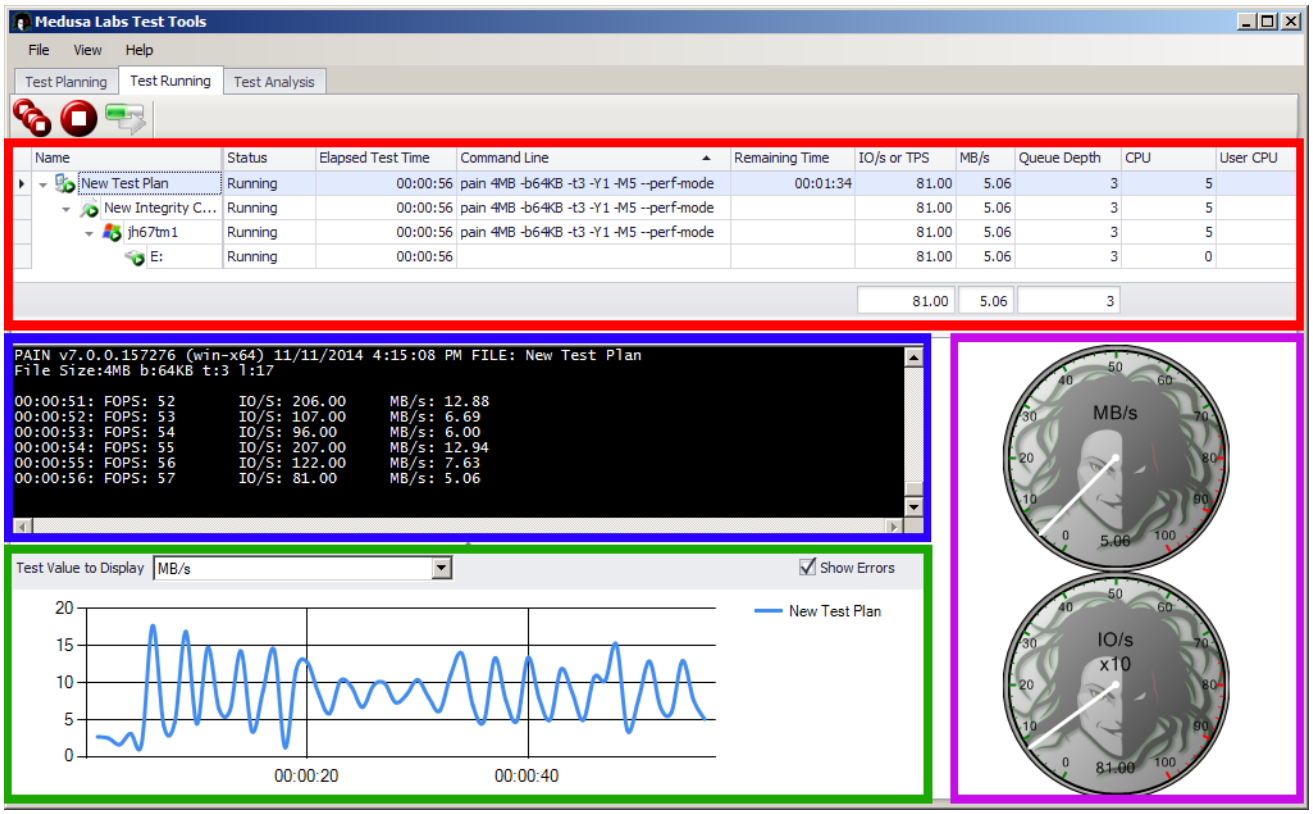
This option converts the configuration into the appropriate CLI configuration. Custom, Integrity, and Performance configurations are converted to Storage CLI configurations. Socket and TCP App Simulations are converted to Network CLI configurations. This option is not available for Storage CLI and Network CLI configurations.

These editors are described in detail in Chapter 3, “Using the Configuration Editors’’ starting on page 61. Refer to the appropriate configuration editor for a description.

Test Running Tab

The **Test Running** tab allows you to display the statistics of the test that you are running and is shown in Figure 29. All tests are removed as soon as they complete.

Figure 29: Sample Text View of the Test Running Tab



The **Test Running** tab has four panes:

- **Test List and Statistics** pane outlined in red.
- **Text View** pane outlined in blue.
- **Graph View** pane outlined in green.
- **Speedometers** pane outlined in purple.

In addition to the four panes, there are three buttons associated with the **Test Running** tab near the top of the tab. These buttons are shown in Figure 30 and described in Table 4.

Figure 30: Test Running Tab Buttons

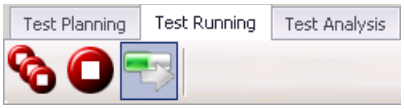
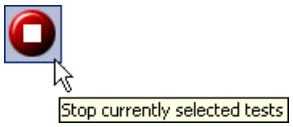


Table 4: Test Running Tab Buttons

	The **Stop all testing** button stops all running tests
	The **Stop currently selected tests** button stops the selected test.

	The **Move to the next test** button ends the test that is currently running and proceeds to the next test in the test plan.
	
	When the Planning Group level is selected:
	• The **Move to the next test** button ends the test plan that is currently running and proceeds to the next test plan by default.
	• However:
	  • If the planning group setup has the "Run all test plans at the same time" option selected, the **Move to the next test** button will stop testing and exit the whole planning group.
	  • If the planning group only has one test plan, the **Move to the next test** button will stop testing and exit the whole planning group.

Test List and Statistics Pane

All running test plans will be listed in the **Test List and Statistics** pane. The details for each of the tests are shown in columns.

To show or hide the test plan objects such as configuration and targets, click the arrow icons (▼ ▶) at the left edge of the test plan.

- When the icon is an arrow pointing right (▶), the objects are hidden; click the arrow to show the objects.
- When the icon is an arrow pointing down (▼), the objects are shown; click the arrow to hide the objects.

The details displayed in the columns can be managed by right-clicking the row of headings. When this is done, a menu is displayed showing a list of available items that can be displayed in a column. From the menu, select a heading to display or remove the column. When an item is checked, it is displayed in the column.

The following is a list of the available details:

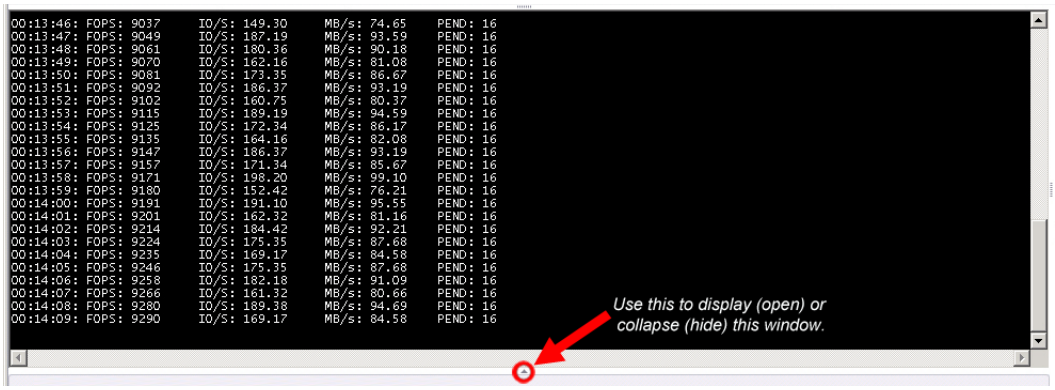
Show All General Columns	Max IO/s or TPS	I/O Timeouts
Hide All General Columns	Max MB/s	License Errors
Name	Max Queue Depth	Open Errors
Status	Min I/O Completion/Response Time (Sec)	Read Errors
Command Line	Min IO/s or TPS	Remove Errors
Elapsed Test Time	Min MB/s	Seek Errors
Remaining Time	Min Queue Depth	Size Errors
Show All Testing Statistic Columns	CPU	Startup Errors
Hide All Testing Statistic Columns	User CPU	Unknown Errors
Avg I/O Completion/Response Time (Sec)	Show All Error Columns	Write Errors
Avg IO/s or TPS	Hide All Error Columns	Show All Final Statistic Columns
Avg MB/s	Automatically Show Columns with Errors	Hide All Final Statistic Columns
Avg Queue Depth	Close Errors	Total Bytes
IO/s or TPS	Data Corruptions	Total Errors
MB/s	Flush Errors	Total File Operations
Queue Depth	Initial Errors	Total I/Os or Transactions
Max I/O Completion/Response Time (Sec)	I/O Halts	

Note: In addition, right-clicking anywhere in the **Test List and Statistics** pane, displays a context menu with several choices that include Stop Test Plan, Stop All Testing, Move to Next Test in Plan, Move All Plans to Next Test, Expand All, Collapse All, and Columns. The Columns choice also provide control of the displayed columns organized by categories.

Text View Pane

The **Text View** pane (shown in Figure 31) displays the results of the selected test plan or its components. As the test runs, this pane shows the running test results at each sample interval. The sample interval was identified in the test plan setup.

Figure 31: Text View Pane

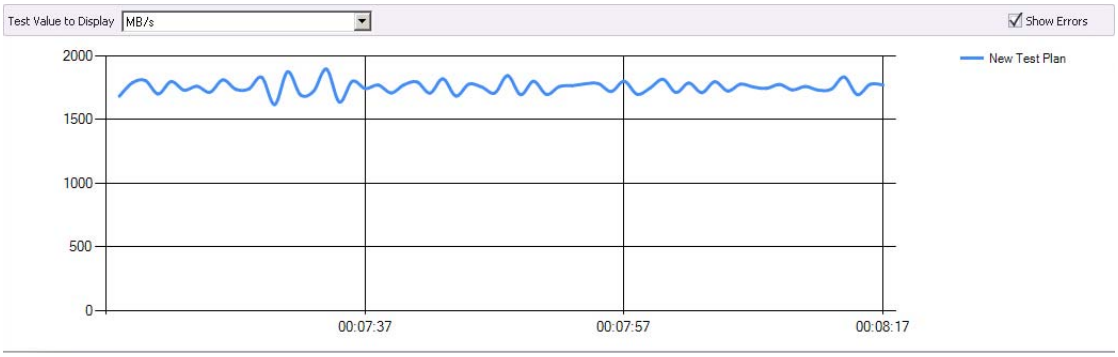


This pane can be opened or hidden by clicking the small arrow icon located at the center bottom of the pane. This icon is shown in the red circle in the illustration above.

Graph View Pane

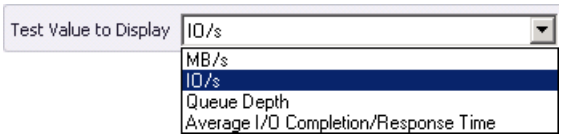
The **Graph View** pane (shown in Figure 32) displays the progression of the test at each sample interval. The sample interval was identified in the test plan setup.

Figure 32: Graph View Pane



You can select the **Test Value to Display** (**MB/s, IO/s**, **Queue Depth**, or **Average I/O Completion/Response Time**) from the dropdown list located above the graph. The dropdown list is shown in Figure 33. Changing this value changes the vertical scale on the graph.

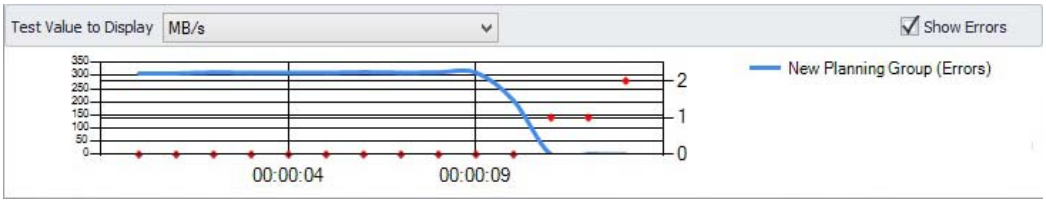
Figure 33: Test Value to Display



At the right side of the **Test Value to Display** list, there is the **Show Errors** check box that, if checked, displays errors as they occur during the testing. If the check box is checked, the test runs normally until an error is detected. Once an error is detected:

- A new Y-axis is displayed on the right edge of the graph to display the number of detected errors. See Figure 34.
- The number of errors at each sample interval point is indicated with a red dot. Note that red dots are also added to interval points that occur prior error at the zero point on the graph.

Figure 34: Graph View Pane with Errors



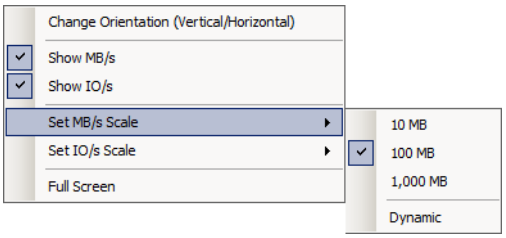
Speedometers Pane

The **Speedometers** pane (shown in Figure 32) displays the real-time speed of the tests. There are two speedometers, one for MB/s and the other IO/s.

Figure 35: Speedometer Pane

The speedometers automatically scale so that when the size of the MLTT window is changed, the **Speedometers** pane adjusts to best fit the screen. Right-click anywhere in the **Speedometers** pane to display additional options as shown in Figure 36.

Figure 36: Speedometer Pane Right-Click Menu



Change Orientation (Vertical/Horizontal) - Select this option to display the speedometer on top of each other (vertical) or side-by-side (horizontal).

Show MB/s – Select this option to display the MB/s speedometer.

Show IO/s – Select this option to display the IO/s speedometer.

Set MB/s Scale – Select this option to choose the intervals displayed on the MB/s scale. You may select 10 MB, 100 MB, 1,000 MB, or Dynamic.

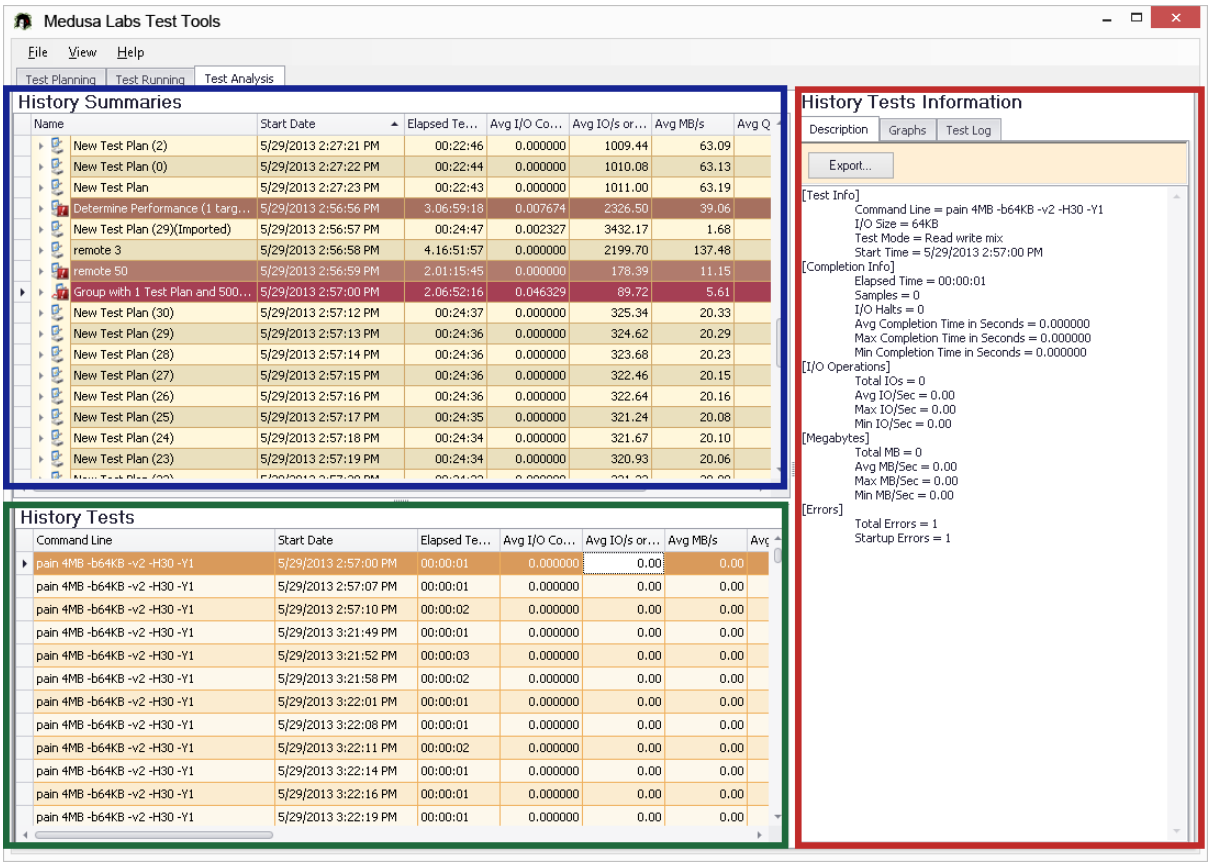
Set IO/s Scale – Select this option to choose the intervals displayed on the IO/s scale. You may select 100 IO/s, 1,000 IO/s, 10,000 IO/s, 100,000 IO/s, or Dynamic.

Full Screen – Select this option if you prefer to display the speedometers using the whole computer screen.

Test Analysis Tab

The **Test Analysis** tab (shown in Figure 37) displays the analysis of outputs of the tests that you have run.

Figure 37: Sample Test Analysis Tab



The **Test Analysis** tab has three panes:

- **History Summaries** pane outlined in blue.
- **History Tests** pane outlined in green.
- History information pane outlined in red. The title of this pane changes based on what test is selected from the two previous panes.
  - When a test summary is selected in the **History Summaries** pane, the title of the pane is: **History Summaries Information**
  - When a test is selected in the **History Tests** pane, the title of the pane is: **History Tests Information**

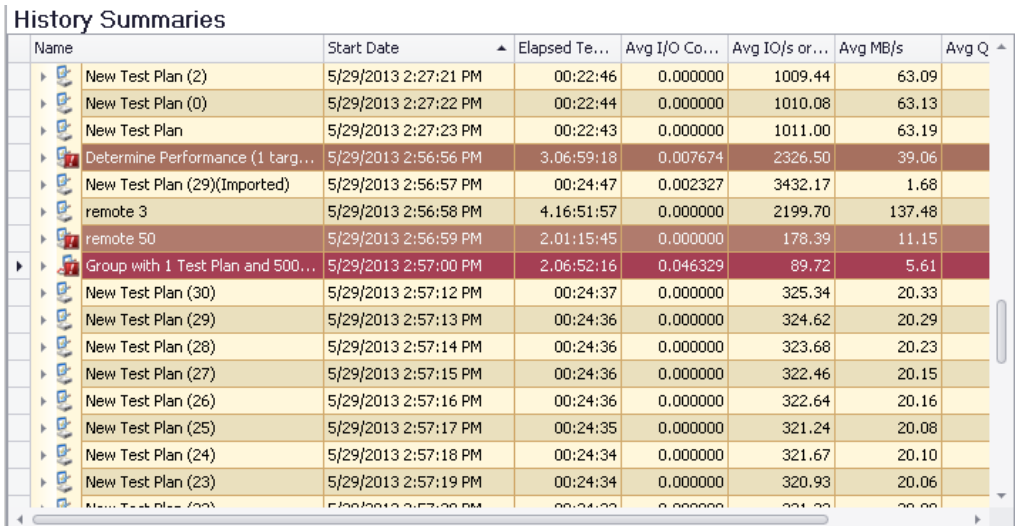
History Summaries Pane

The **History Summaries** pane displays all completed test plans. Specific details about each of the tests is shown in columns.

To show or hide the lower-level objects of a test plan group (such as the test plans, configurations and targets), click the arrow icons (▼ ▶) at the left edge to show (expand) or hide (collapse) the subordinate objects. (Refer to "Test Plan Browser" on page 37 for a discussion regarding test plan group and test plan hierarchy.)

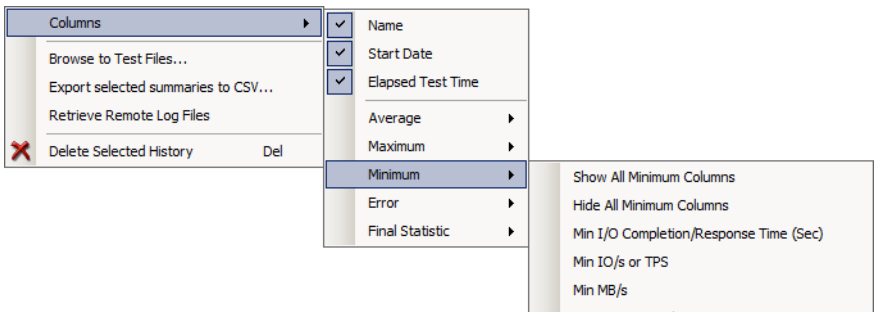
- When the icon is an arrow pointing right (▶), the subordinate objects are hidden; click the arrow to expand the tree and show the objects.
- When the icon is an arrow pointing down (▼), the objects are shown; click the arrow to collapse the tree and hide the objects.

Figure 38: History Summaries Pane



By default, the column information for each test includes Name, Start Date, Elapsed Test Time, Avg (average) I/O Completion/Response Time, Avg I/Os or TPS, Avg MB/s, Avg Queue Depth, and Total Errors. However, if you right-click in the pane, a menu is displayed that will allow you to select from several parameters to add as columns to the table. See Figure 39.

Figure 39: History Summaries Pane Right-Click Menu



In the columns selection, there the Name, Start Date, and Elapsed Test Time selections that you can select to show or hide that column in the results. There is also five groups: Average, Maximum, Minimum, Errors, and Final Statistic that can be selected to show or hide results in these categories.

From each of these groups, you can elect to show or hide all results from within the group or show or hide individual results within the group. In the Errors group, you also have the option of automatically showing error columns. This only shows an error column of a specific type if there are errors of that type.

Using the right-click menu also allows you to browse for test files, export selected summaries as .csv files, retrieve remote log files, and delete the selected history file.

Note: Right-clicking over a column displays all of the columns selections vertically without displaying the groups listed above. This may be used to save key strokes.

History Tests Pane

The individual command line commands of the selected test are displayed in the **History Tests** pane.

By default, the column information for each command line test includes Command Line, Start Date, Elapsed Test Time, Avg I/O completion/Response Time (Sec), Avg I/Os or TPS, Avg MB/s, Avg Queue Depth, and Total Errors. Note that the Name is not included in the default view.

Figure 40: History Tests Pane

History Tests

Command Line	Start Date	Elapsed Te...	Avg I/O Co...	Avg IO/s or...	Avg MB/s	Avg
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 2:57:00 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 2:57:07 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 2:57:10 PM	00:00:02	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:21:49 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:21:52 PM	00:00:03	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:21:58 PM	00:00:02	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:01 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:08 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:11 PM	00:00:02	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:14 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:16 PM	00:00:01	0.000000	0.00	0.00	
pain 4MB -b64KB -v2 -H30 -Y1	5/29/2013 3:22:19 PM	00:00:01	0.000000	0.00	0.00	

As with the **History Summaries** pane, if you right-click in the **History Tests** pane, a menu (shown in Figure 39) is displayed that will allow you to select from several parameters to add as columns to the table. In the columns selection, there the Name, Command Line, Start Date, and Elapsed Test Time selections that you can select to show or hide that column in the results. There is also five groups: Average, Maximum, Minimum, Errors, and Final Statistic that can be selected to show or hide results in these categories.

From each of these groups, you can elect to show or hide all results from within the group or show or hide individual results within the group. In the Errors group, you also have the option of automatically showing error columns. This only shows an error column of a specific type if there are errors of that type.

Using the right-click menu in the **History Tests** pane, you may select summaries to be exported to a .csv file using the **Export all summaries to CSV...** or **Export selected summaries to CSV...** selections.

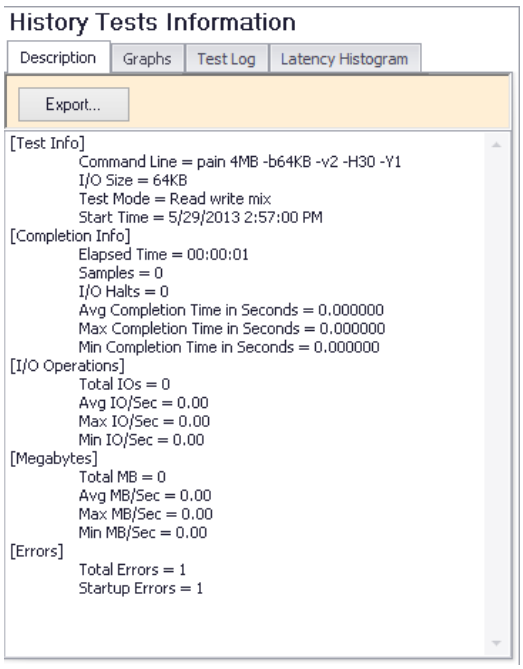
Note: Right-clicking over a column displays all of the columns selections vertically without displaying the groups listed above. This may be used to save key strokes.

History Information Pane

The title of this pane changes based on what test is selected from the two previous panes.

- When a test summary is selected in the **History Summaries** pane, the title of the pane is: **History Summaries Information**
- When a test is selected in the **History Tests** pane, the title of the pane is: **History Tests Information**

Figure 41: History Information Pane (History Summaries Information Version)



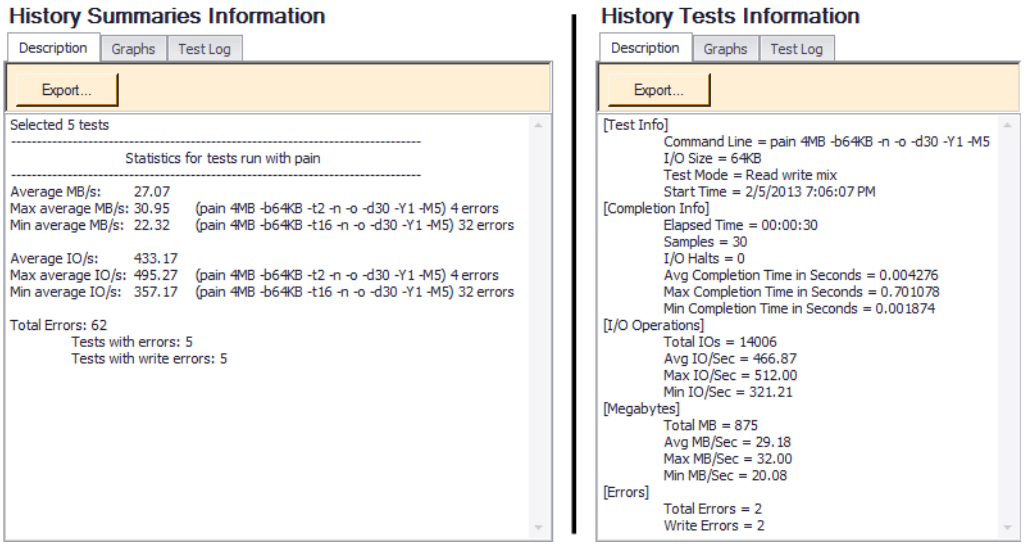
The History Information pane has four tabs:

- **Description** (See “Description Tab” on page 54.)
- **Graphs** (See “Graphs Tab” on page 55.)
- **Test Log** (See “Test Log Tab” on page 59.)
- **Latency Histogram** (See “Latency Histogram Tab” on page 60.)

Description Tab

The **Description** tab provides a summary of the selected test from the **History Summaries** pane (shown on the left side of Figure 42) or the selected command line(s) from the **History Tests** pane (shown on the right side of Figure 42).

Figure 42: Description Tab Examples



In addition, the **Export...** button saves the description as a file in .prf format. The .prf files can be used with the prfgrab tool (provided with Medusa Labs Test Tools Suite) to help prepare performance reports.

The performance summary gets exported with its designated native extension of .prf. While it is just a text file, the tools use various file extensions to identify their function (.log, .bad, .dbg, etc.) You can create a script or use the sample script provided to run a variety of test cases that will result in the creation of a uniquely named .prf file for each test case. You can then use the prfgrab tool to consolidate all those .prf files into a .csv file for sorting and graphing in Microsoft Excel.

Graphs Tab

The **Graphs** tab provides a graphical representation for the test(s) that you have selected in either the **History Summaries** or the **History Tests** panes. By default, the graph shows a line graph of the average IO/s and average MB/s values, however you have the option to change these views using the Graphing Options. Refer to "Graphing Options" on page 55 for additional information.

The graphing algorithm attempts to be as robust as possible when test groups, test plans, and targets (in the **History Summaries** pane) and tests (in the **History Tests** pane) are selected. However, there may be an extreme range of variables in selections that you are able to make. The graphing algorithm makes a best effort to graph something meaningful by looking at the command lines.

The algorithm for graphing multiple **History Tests** selections tries to group the tests in the horizontal x-axis by pain/maim, write/read, IO size, and thread count, elapsed time, and queue depth.

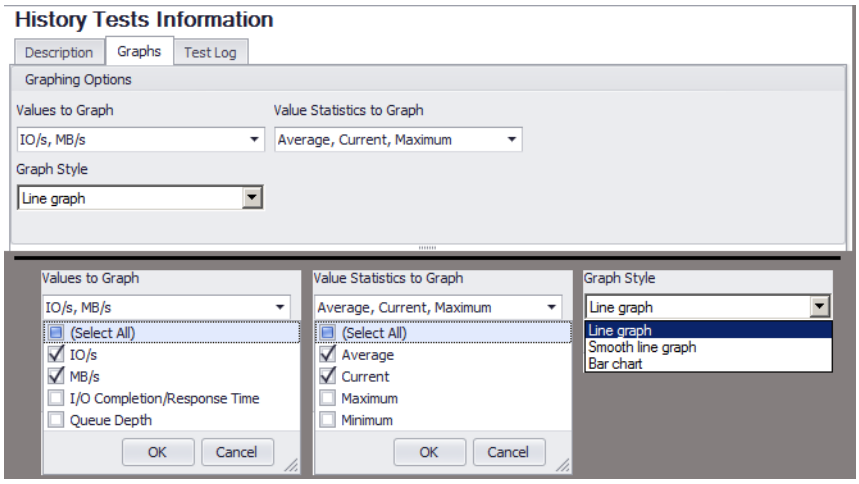
If there are a lot of command line differences between the tests and they cannot be fit in those groups then the graphing algorithm uses the best common thread between the tests that it can determine.

Note: When making graph selections, it is important to select items that make sense to graph. This is best determined by reviewing the graph's x-axis for appropriateness to providing helpful information.

Graphing Options

The **Graphing Options** area provides three sets of options to graph, **Values to Graph**, **Value Statistics to Graph**, and **Graph Style**.

Figure 43: Graphing Options Area With All Available Options Shown Below



The **Values to Graph** dropdown list allows you to choose IO/s, MB/s, I/O Completion/Response Time, and Queue Depth. When one of these choices are added (or removed), a graph is added to (or removed from) the display. The default selections are IO/s and MB/s.

The **Value Statistics to Graph** dropdown list allows you to choose Average, Current, Maximum, and Minimum. When one of these choices are added (or removed), a line/bar is added to (or removed from) each of the graphs. The default selections are Average and Current.

The **Graph Style** dropdown list allows you to Line graph, Smooth line graph, or Bar graph to view the data as it best fits your needs. The default selection is Line graph.

- **Line graph** – provides a point-to-point graphing of the measured data.
- **Smooth line graph** – provides a smoothing of the line graph to provide a more aesthetic view of the line graph. When the graph is smoothed, the high and low values may have a slight loss of accuracy.
- **Bar graph** – shows the measured values relative to each other in standard bar graph format.

Read/Write, Read, and Write Tabs

The configuration editors have an **I/O Payload** tab that allows you to select the Read/Write Mix setting: either **Read/Write**, **Read Only**, **Write Only**, or **Specify Custom Read/Write Mix**. When you select a test planning group in the **History Summaries** pane that ran using multiple configurations with different Read/Write Mix settings, their graphs will be grouped showing their Read/Write Mix on a tab labeled **Read/Write**, **Read**, or **Write**.

Pain and Maim Tabs

When you select a test planning group in the **History Summaries** pane that ran using two different tool configurations (one using Pain and one using Maim), their graphs will be grouped by their tools one showing a **Pain** tab or a **Maim** tab.

Burst and Static Tabs

When Maim is selected in a configuration, by default, the Queue Depth uses burst queuing. The Queue Depth also has the **Keep Queue Depth Static** check box. If **Keep Queue Depth Static** is selected, continuous queuing is used.

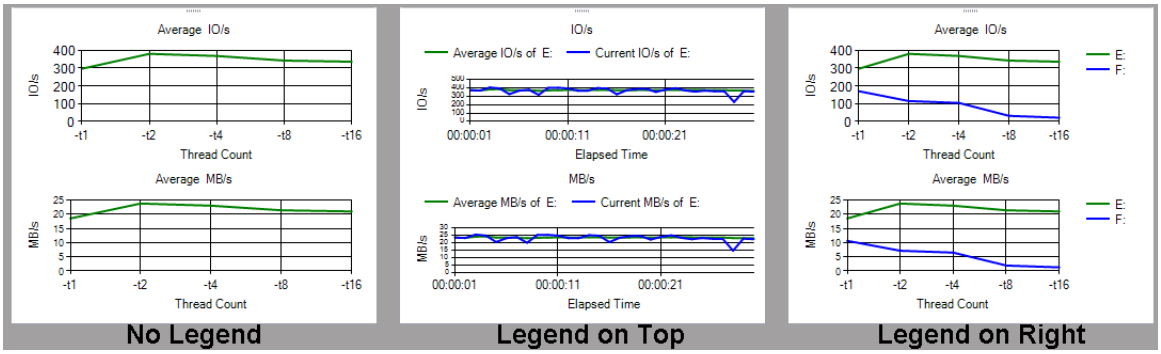
When you select a test plan in the **History Summaries** pane that uses both types of queuing (burst and continuous), their graphs will be grouped by their queuing type with burst queuing displayed on the **Burst** tab and continuous queuing displayed on the **Static** tab.

Graph Legends

When there is only one plot on a graph, no legend is displayed. When there is more than one plot on a graph, a legend is displayed as described below. See Figure 44.

- If multiple IO sizes are plotted, the legend is display at the right of the graph.
- If multiple statics are plotted, the legend is displayed above the graph.

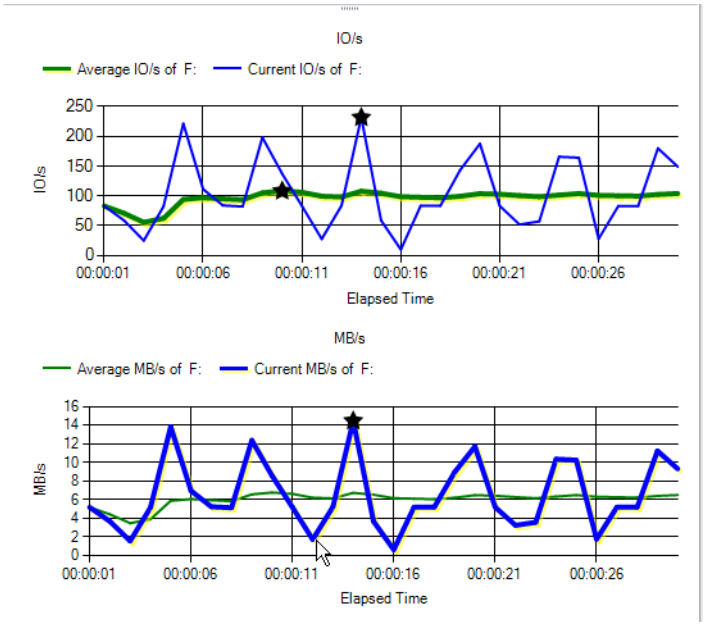
Figure 44: Graphs Legends



Displaying the Highest Value on a Plot

The highest value on a plot can be identified by clicking any point on a plot or by click any legend. Once the plot or the legend is clicked, a start is inserted on the plot showing the highest value or point. If you have multiple plots on a graph, you can click once on each plot to show the highest value of each. In Figure 45, the upper graph shows both plots displaying their highest value and the lower graph shows one plot after it was clicked.

Figure 45: Displaying the Highest Value on a Plot

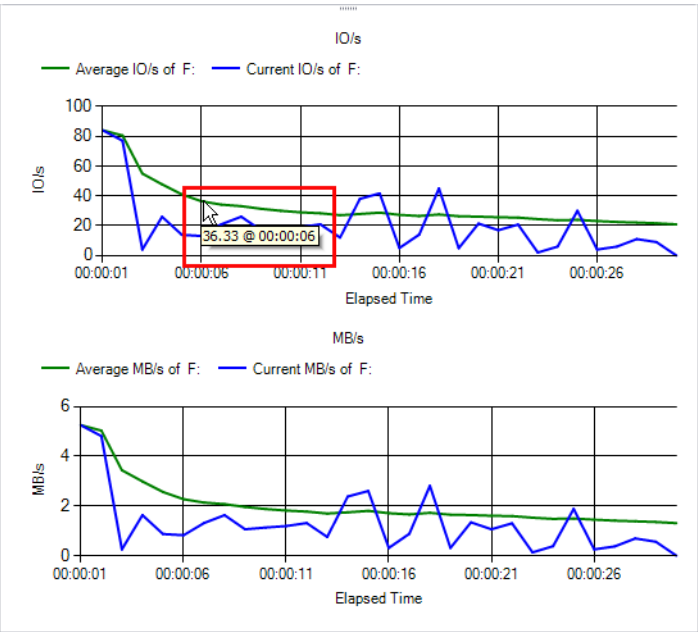


Displaying Values on Line Graphs

You can display the value at any point on a line graph or a smooth line graph by moving the cursor to a point on the plot and hovering over that point. A text box showing the values is displayed.

As shown in Figure 46, hovering your cursor over a point on the plot displays the values of the graph's coordinates at that point.

Figure 46: Value Displayed by Hovering the Mouse Cursor Over the Plot



Zooming In/Out

For graphs with Elapsed Time as the horizontal axis, you can zoom in to view the graph at a higher resolution. Click the graph to give it focus, then zoom in (or out) using the:

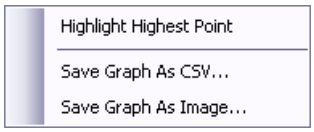
- mouse's scroll wheel
- keyboard's +/- keys or the "q"/"w" keys

When the information becomes too wide to be displayed without scrolling, a scroll bar is provided at the bottom of the graph.

Saving the Graphs

Right-clicking a graph displays the image shown in Figure 47. You can save the graph as a Comma-Separated Values file (.csv) to be used in a spreadsheet or as an image file. When you select to save the graph as an image, you may save it as a Portable Network Graphic (.png), as a JPEG (.jpg), or as bitmap (.bmp) image.

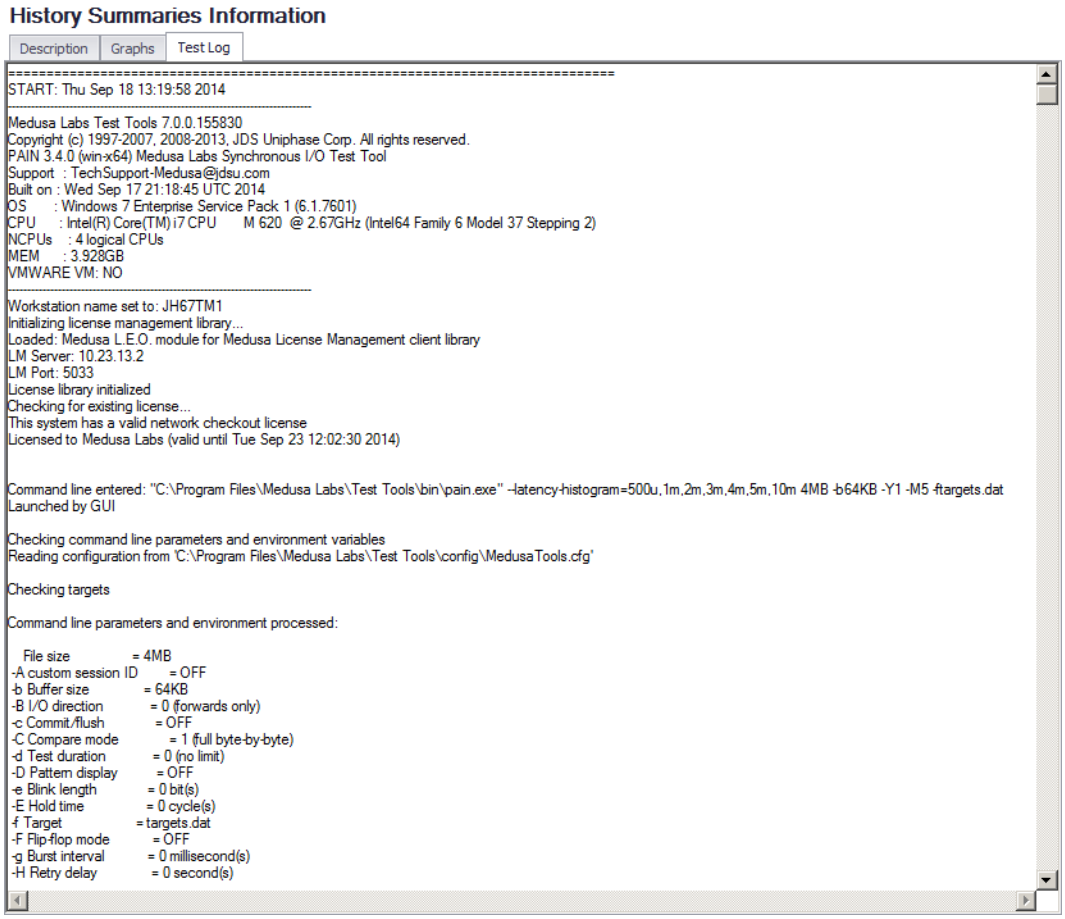
Figure 47: Saving the Graph



Test Log Tab

For the **Test Log** tab, data is provided for selections made in the **History Tests** pane only. If no selection is made in the **History Tests** pane, the data from the first one will be automatically displayed. This tab displays exactly what would be shown on your display by running the command line on your computer. Until other tests are selected, the first test will be displayed by default.

Figure 48: Test Log Tab



Latency Histogram Tab

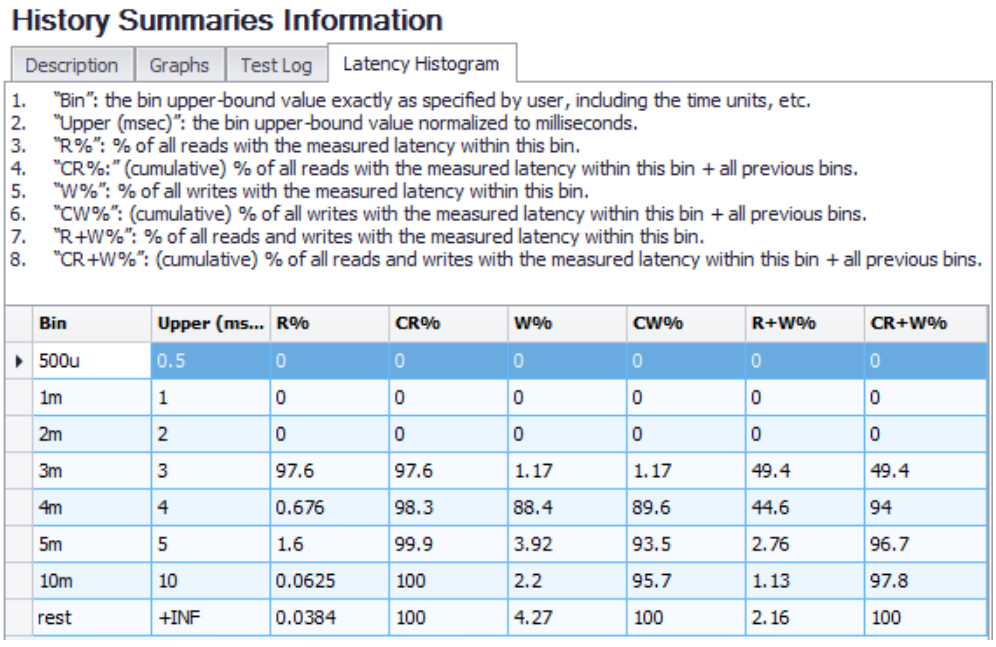
Latency histogram collects latency histogram per target. The collection bins are specified when using the Custom configuration editor (see page 67). The bins are sorted by the magnitude of the upper bound values, and the range of each bin is constructed such that the upper bound is as specified and the lower bound is the upper bound of the previous bin. .

Important: The Latency Histogram table (with its data) is only displayed when the drive (of a configuration) utilizing Latency Histogram is selected in the **History Summaries** pane. If no drive has been selected, the **Latency Histogram** tab advises you to select a drive.

Note: The Latency Histogram table will not be displayed when the test is running and it will be available for viewing after all tests in the Test Group or Test Plan are completed.

The **Bin** column lists the upper-bound of the range as you give it in the command line. The **Upper (msec)** column is the upper bound value normalized to milliseconds. As an example, a 10us bin would be normalized as 0.01 while 5s would be normalized as 5000. The other columns, **R%**, **W%**, and **R+W%** display the percentage of Reads Write, or Read/Write operations with measured latency that are within the bin; while **CR%**, **CW%**, and **CR+W%** display the cumulative value of the percentage for Read, Write, or Read/Write operations with measured latency through each bin. The last row, **rest**, is a bin that is added for operations with latency greater than the largest specified bin. **INF** (for infinity) is inserted in this row as this bin cannot be normalized.

Figure 49: Latency Histogram Tab



Bin	Upper (ms...	R%	CR%	W%	CW%	R+W%	CR+W%
500u	0.5	0	0	0	0	0	0
1m	1	0	0	0	0	0	0
2m	2	0	0	0	0	0	0
3m	3	97.6	97.6	1.17	1.17	49.4	49.4
4m	4	0.676	98.3	88.4	89.6	44.6	94
5m	5	1.6	99.9	3.92	93.5	2.76	96.7
10m	10	0.0625	100	2.2	95.7	1.13	97.8
rest	+INF	0.0384	100	4.27	100	2.16	100

Chapter 3

Using the Configuration Editors

In this chapter:

Using the Configuration Editors within the GUI

This chapter provides detailed information for editing configurations. Configurations will normally be edited when creating new configurations in the Configurations area (page 32) or when editing existing configurations in the Test Plans area (page 44). Each of these areas has the New Configuration button described below. Each of the configuration editors are described in the following pages.

Note: Configuration Editors are accessible using a variety of methods in the Configurations area and the Test Plans area.

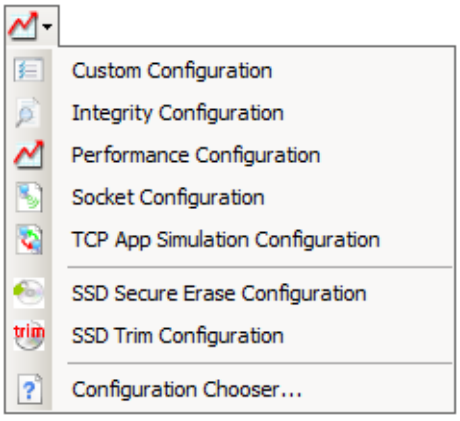
To access configurations from the Configurations area, refer to "Configurations Area" on page 32.

To access configurations from the Test Plans area, refer to "New Configuration Button" on page 35.

New Configuration Button

This button opens the following drop-down list of configurations. Selecting one of these configurations creates a new blank configuration in the **User Configurations** folder that is located in the area below the button. You can also select the **Configuration Chooser** from the menu to open the **Configuration Chooser** window.

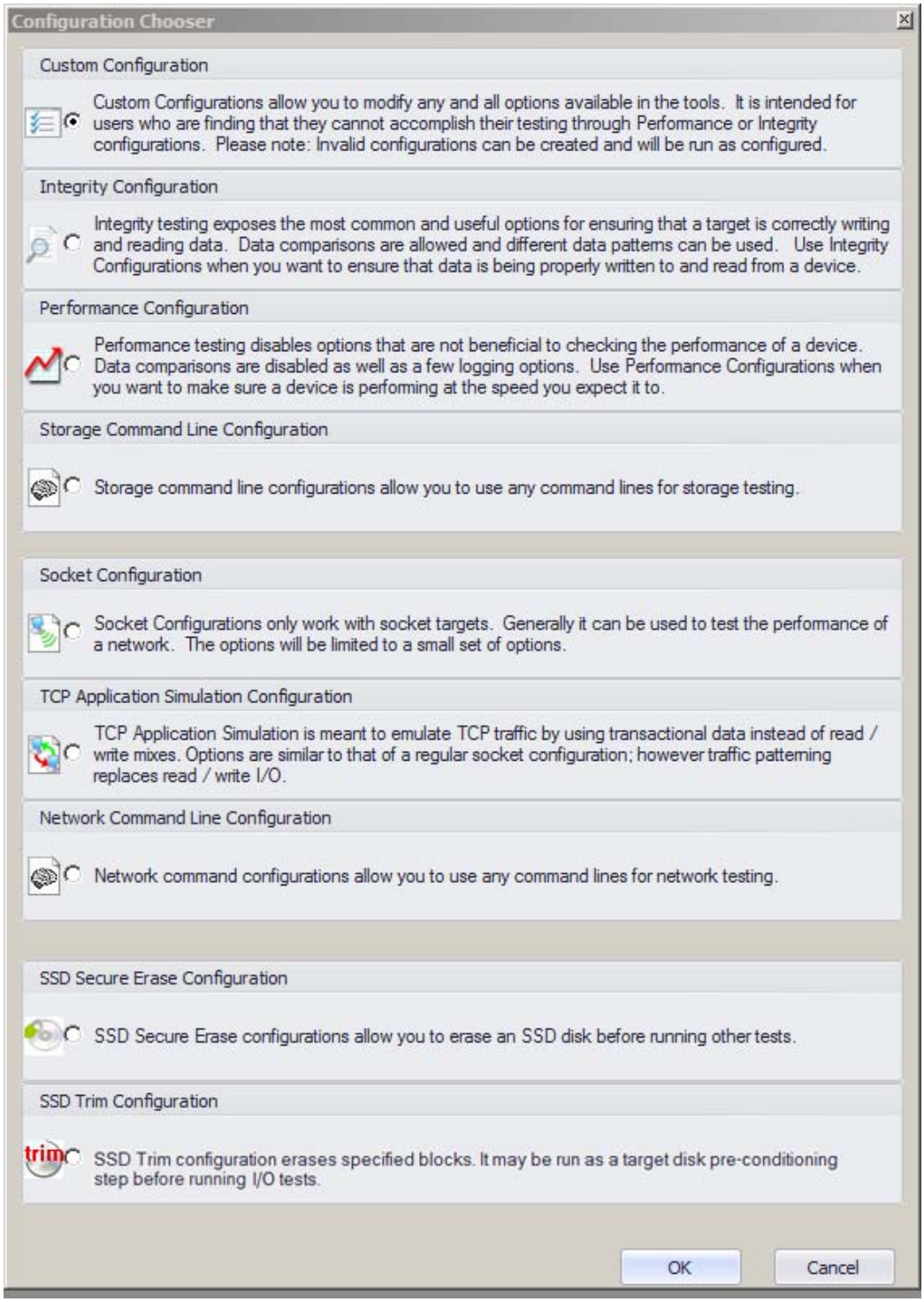
Figure 50: Create New Configuration Button



Configuration Chooser Window

The **Configuration Chooser** window allows you to select the configuration for the test that you want to run.

Figure 51: Configuration Chooser Window



Select the radio button of the configuration you want to use and click **OK**.

The following configurations are available:

- **Custom Configuration**
  Custom Configuration allows you to modify any and all options available through the tools. It is intended for users who are finding that they cannot accomplish their testing through Performance and/or Integrity configuration. An invalid configuration can be created and will be run as configured. For more information on how to edit custom configuration settings, see “Custom Configuration Editor” on page 66.
- **Integrity Configuration**
  Integrity Configuration testing exposes the most common and useful options for ensuring that a target is correctly writing and reading data. Data comparisons are allowed and different data patterns can be used. Use Integrity Configuration when you want to ensure that data is being properly written to and read from a device. For more information on how to edit integrity configuration settings, see “Integrity Configuration Editor” on page 80.
- **Performance Configuration**
  Performance Configuration testing disables options that are not beneficial to checking the performance of a device. Data comparisons are disabled as well as a few logging options. Use Performance Configuration when you want to make sure a device is performing at the expected speed. For more information on how to edit performance configuration settings, see “Performance Configuration Editor” on page 91.
- **Storage Command Line Configuration**
  Storage Command Line Configuration supports pain and maim commands. For information on using the Storage Command Line configuration editor, see “Storage CLI Configuration Editor” on page 97.
- **Socket Configuration**
  Socket Configuration only works with socket targets. Generally this configuration can be used to test the performance of a network. The options will be limited to a small set of options. For more information on how to edit socket configuration settings, see “Socket Configuration Editor” on page 98.
- **TCP Application Simulation Configuration**
  TCP Application Simulation Configuration is meant to emulate TCP traffic by using transactional data instead of read/write mixes. Options are similar to that of a regular socket configuration; however, traffic patterning replaces read/write I/O. For more information on how to edit TCP application simulation configuration settings, see “TCP App Simulation Configuration Editor” on page 109.
- **Network Command Line Configuration**
  Network Command Line Configuration supports socket commands. For information on using the Network Command Line configuration editor, see “Network CLI Configuration Editor” on page 118.
- **SSD Secure Erase**
  SSD Secure Erase Configuration allows you to erase an SSD disk before running other tests. For information on using the SSD Secure Erase configuration editor, see “SSD Secure Erase Configuration Editor” on page 119.
- **SSD Trim**
  SSD Trim Configuration allows you to erase unused blocks to pre-condition a target SSD disk before running other tests. For information on using the SSD Secure Erase configuration editor, see “SSD Trim Configuration Editor” on page 121.

Configuration Editors

You can modify the different test configurations through the configuration editors. Each of the configuration editors is described in the following sections:

Test a Range Controls

On the I/O Payload tab of each configuration editor, the **Testing Threads**, the **Queue Depth**, and the **Testing Sizes** areas allow you to select a range for testing. This “Test a Range” option allows you the set specific **Start** and **End** values for these parameters.

It also provides you **Adding** and **Multiplying** settings. Use the **Adding** and **Multiplying** to define how the configuration gets from the start value to the end value.

Using **Testing Threads** as an example, if you have the thread count set to start at 1 and end at 64:

If you select Add by 1, you will get thread counts of:	**If you select Multiply by 2, you will get thread counts of:**
pain -t1 pain -t2 pain -t3 pain -t4 … pain -t63 pain -t64	pain -t1 pain -t2 pain -t4 pain -t8 pain -t16 pain -t32 pain -t64

Custom Configuration Editor

The Custom configuration editor allows you to edit available options when you want to create unique testing configurations.

To open the editor, double-click the custom configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: Some of the options on the editor may be grayed out or not displayed based on the methodology selected or other option dependencies. The editor opens in the same mode, depending on the mode when it was closed.

The editor has seven tabs for specifying testing parameters:

- General (See page 67)
- I/O Payload (See page 68)
- I/O Behavior (See page 72)
- Advanced I/O (See page 75)
- Patterns (See page 75)
- Comments (See page 79)
- Command Lines (See page 79)

The description of each of these tabs and its parameters in the following pages.

General Tab

The **General** tab shows the following settings for this configuration editor.

Startup Options –

Initial Sample Delay – Sets the delay time to wait before starting the testing. This delay allows devices the allotted time to get setup. The **H** (Hour), **M** (Minute), and **S** (Second) numeric spinners allow you to delay the start of the testing up to 23 hours, 59 minutes, and 59 seconds.

Throughput Limitation – sets the limitations for the maximum I/O throughput

Infinity – sets the maximum I/O throughput to have no limitations.

Every Target – sets the **Max I/O Throughput** value to apply to every target.

Every Thread – sets the **Max I/O Throughput** value to apply to every thread.

Max I/O Throughput – provides the maximum I/O throughput limitation value that is applied to every target or thread.

External Application – runs any application of your choosing after every test has been run. For example, if you want to run an independent application to collect data (such as a log file) from a device, you can use this to start the application.

The **Run External Application After Test** check box must be selected before you can select the application and wait times.

Application allows for you to browse to and select the desired application.

Wait for External Application allows you to input a period to wait for the application to start to run. Tests tools will continue testing once the external application begins running.

Latency Histogram – collects latency histogram per target.

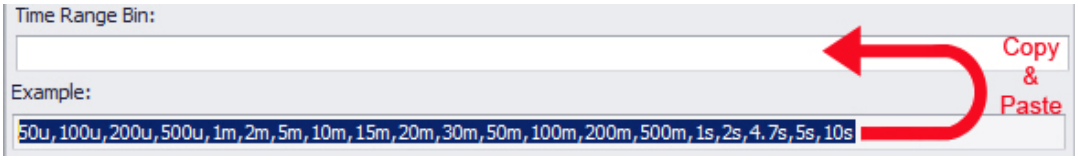
The collection bins are specified by entering the upper bound of each bin as a comma-separated list in the **Time Range Bin** text box. The list is sorted by the magnitude of the upper bound values, and the range of each bin is constructed such that the upper bound is as specified and the lower bound is the upper bound of the previous bin.

Upper bounds may be specified as a floating point value (e.g. "0.5" or "4.5").
The time unit suffix may be used:

'n' for "nano" 'u' for "micro" 'm' for "milli" 's' for "seconds"

If no time unit suffix is given, 'm' for "milli" is assumed. .

Note: A quick and easy method of specifying the upper bound of each bin in the **Time Range Bin** text box is by copying the values in the Example and pasting them into the text box.



You can then edit the values in the **Time Range Bin** text box if you choose.

The Latency Histogram tab (see page 60) displays the collected histogram data.

Steady State

Steady State determines the steady state for a target across five consecutive test runs. With the test plan set to run indefinitely or several times (5 times or more), when steady state is achieved, the test plan will be stopped when the current test iteration completes. If the test plan is part of a planning group, the next test plan in the group will begin.

If steady state is not achieved during the specified number of test runs, the test plan will complete its last iteration and testing is terminated. Subsequent test plans in the planning group are ignored. If you select to run the test plan indefinitely and steady state is not achieved, you will need to stop the test manually.

Select the **Check for Steady State** check box to enable this feature. Once the check box is selected, you can set the following options:

Tag text box allows you to enter an arbitrary string that is not 'r', 'iops', 'mbps' or 'lat' which can be used to uniquely identify a steady state testing case. This tag will be added to the steady-state.csv file name.

Tracking Variable group allows you to select one of the Tracking radio buttons and set the Deviation percentages.

Tracking radio buttons:

IOPS Tracks IOPS for steady state. (default value)

MBPS Tracks MBPS for steady state.

IO Latency Tracks I/O latency for steady state.

Deviation percentages:

% Range Deviation: Allowed deviation of minimum and maximum tracked values from the average. The default value is 20%.

% Slope Deviation: Allowed deviation of minimum and maximum points in a best linear fit line through the tracked values. The default value is 10%.

I/O Payload Tab

The **I/O Payload** tab shows the basic parameters for a Test Tool test, such as the testing style (synchronous or asynchronous), testing threads, queue depth, I/O operation size, read/write mix, I/O type, I/O marking and signing, and logging level.

Performance – Select the **Enable Performance mode** check box to enable the performance mode. This mode increases the speed of testing by optimizing the use of internal memory buffers. When this check box is selected, the following settings are automatically set:

Table 5: Performance Mode Settings

GUI Setting	Command Line Setting
I/O Marking and Signing is set to **No I/O Markings** No I/O signatures are applied to each sector of every write.	`-u` (page 177)
Data Compare Mode is set to **Disable Data Comparisons** Data comparisons are turned off.	`-n` (page 177)
I/O Behavior is set to **Keep File Handles Open Between I/Os** Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP.	`-o` (page 166)
Use Pattern Reversals check box is not selected (cleared) Leaves the data patterns reversal after each FOP (forward, then backward) turned off.	`-N` (page 174)

Testing Style – Select **Synchronous (Pain)** or **Asynchronous (Maim)**.

When **Synchronous (Pain)** is selected, set the **Testing Threads** area. Also select the appropriate SCSI passthrough option from the drop-down menu. The options are listed below:

- SCSI Passthrough Off
- READ/WRITE 10
- READ/WRITE 10 + FUA (Forced Unit Access)
- READ/WRITE 16
- READ/WRITE 16 + FUA (Forced Unit Access)

When **Asynchronous (Maim)** is selected, set the **Testing Threads** area. Also set the **Queue Depth** area.

Testing Threads – Set the threads for testing. Each thread executes a single I/O at a time, with each thread starting at a different base offset. The number of threads successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

You can also set a range of threads to test by selecting the **Test a Range of Threads** check box. Set the **Thread Count Start** and the **Thread Count End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the tested threads through the specified range from the start to the end. Refer to “Test a Range Controls” on page 65 for more information about the **Adding** and **Multiplying** selections.

Queue Depth – (displayed for Asynchronous (Maim) only) Set the queue depth.

The queue depth is the maximum number of current I/Os to execute in a single worker thread. The value of queue depth successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the queue depth by entering the value or clicking the **Queue Depth** numeric spinner.

You can also set a range for the queue depth by selecting the **Test a Range of Queue Depths** check box. Set the **Queue Depth Start** and the **Queue Depth End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

The **Queue Depth** has two additional options, **Keep Queue Depth Static** and **Strict Sequential**.

Select **Keep Queue Depth Static** to add `-m16` option to the command line to use continuous queuing. If this option is not selected, by default burst queuing will be used.

Select **Strict Sequential** to add the -m1 option to the command line which will make the test have continuous queuing and strict sequential access.

Testing Sizes – Select the **I/O Operation Size** to be used for testing.

You can set the I/O operation size by entering the value or clicking the **I/O Operation Size** numeric spinner. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Edit the number or click the numeric spinner. Select the unit by clicking the drop-down button.

You can also set a range for the I/O operation size by selecting the **Test a Range of I/O Operation Sizes** check box. Set the **I/O Operation Size Start** and the **I/O Operation Size End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Additional **Testing Sizes** settings include:

Base File Size on I/O Operation Size – Select this option to use the **I/O Operation Size** as basis for the testing area. For synchronous tests (pain), the file size is equal to the block size. For asynchronous tests (maim), the file size is the block size multiplied by the queue depth.

Specify Testing Area – Select this button and specify the file size or disk area to use per worker thread. The size can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Test Using the Entire Target – Select this check box to use the entire target for the test. This is not applicable for file system and memory testing.

Read/Write Mix – Select the read or write mode of the test.

Cycle Through the Read / Write Modes – Select this check box to cycle through the various read/write modes. When this check box is selected, the following Read/Write mix settings are not applicable and they are not displayed.

Read / Write – Select this option to have a balance of read and write testing.

Read Only – Select this option to have a read-only test. If this option is selected, the **Force Initial Write** and **Do Not Perform Initial Write** radio buttons and will be available.

Force Initial Write – Performs a Write I/O once followed by continuous Reads.

Do Not Perform Initial Write – Performs pure read-only I/Os. This also disables data comparison automatically.

Write Only – Select this option to have a write-only test.

Specify Custom Read/Write Mix – Setting the slider in the middle gives 50%/50% chance to Read/Write to be the next IO. This results in a random mix of reads/writes.

Use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the write operations, while sliding it to the right increases the read operations.

- Sliding to the left most makes it a **Write only** test, where it writes a test pattern but does not read it.
- Sliding to the right most makes it a **Read only** test, where it only returns the data that exists in the file or device area.
- Setting the slider to the middle makes it a **Read/Write** test, where it repeatedly writes a test pattern and then reads it back.

I/O Type – Choose from **Forwards Only**, **Alternate Between Forwards and Backwards**, **First FOP Forwards, Rest Backwards**, **Backwards Only**, and **Custom Mixture**.

When you select **Custom Mixture**, use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the percentage of randomness, while sliding it to the right increases the sequential operation.

This slider works in unison with the **% Random** and the **% Sequential** boxes so that the sum of both values is 100 percent. As the slider is moved, the values in the **% Random** and the **% Sequential** box values change to reflect the slider position.

Likewise, when either the **% Random** or the **% Sequential** boxes are changed by entering a value or using the numeric spinner, the other box and the slider are adjusted to reflect the change.

I/O Marking and Signing – Select the I/O Marking and Signing option from the drop-down list. For details on I/O signatures, refer to Appendix D, “I/O Signatures”.

No I/O Markings – No I/O signatures are applied to each sector of every write.

Uniquely Mark I/O (Default) – I/O signatures are applied to each sector of every write, unless disabled.

Add Time Stamps to I/O – Select this option to add timestamp for the I/O event as part of the I/O signature. Timestamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Add Time Stamps in Milliseconds to I/O – Select to choose a time stamp for the test session. Select this option to add timestamp (in milliseconds) for the I/O event as part of the I/O signature. Time stamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Override Session Id – Select this to override the default session ID and specify your own. The default session ID is a semi-unique field in the I/O signature that created using the hex values of the last two characters of the target name. It is way to help identify the host and target when you are debugging.

Logging Level – Select the type of logging you want for the configuration to indicate the level of information to be posted to the log file. The default option posts the maximum amount of information which is helpful for analyzing errors.

Standard logfile generation/output – Default option, includes detailed headers and console performance output.

Outputs to logfile in test performance format (minimal logging) – Removes headers and logs performance output only.

No outputs to logfile, minimal screen outputs, PRF log summary – No .log file generated, and logs minimal screen output.

Disable CSV log – Standard logging, but .csv file will not be created.

Single line output with system name, performance, and errors – Includes system name and other details on single output lines for easier importation or parsing.

Disable completion statistics in PRF file – Disables completion calculations and output. In IOPS intensive tests where the CPU is heavily taxed, using this option may result in a slight performance gain.

Enable logging of informational events in Windows event log – This option will send informational output to Windows event log, such as test start and stop details.

Command Line – As options are selected, the equivalent command line settings appear in the textbox. See Chapter 4, “Using the Command Line Switches’’ for details of the command line settings.

I/O Behavior Tab

The **I/O Behavior** tab allows you to specify data comparisons, set I/O behavior, setup triggering options based on test results, use non-default target offsets, and setup error handlers.

Data Compare Mode – Click the drop-down list to choose a data comparison mode. A byte-for-byte data comparison of write and read data will catch any possible data corruption. Data comparison is usually recommended except in special cases, such as when the overhead of full buffer comparisons decreases the I/O throughput to the target.

Disable Data Comparisons – Data comparisons are turned off.

Full byte-for-byte Comparison – Each byte is checked for integrity (default value).

Signature Comparison Only – (2-3 words every 512 bytes) Checks only the unique I/O signature in the data buffer. This substantially reduces processor utilization in the host system.

Session ID Comparison Only – (16 bit ID at 2nd word every 512 bytes) Compares only the session ID used in the data signature. The session ID is generated from the host and target names. This option can be used with the “Override default session ID” option and is usually used in multi-initiator setups as a quick way of verifying that an initiator’s storage has not been written to by another initiator.

Session ID Comparison, Followed by Full – This option is a combination of the **Session ID Comparison Only** check with a full data comparison check. This is another method used in multi-initiator setups, typically used when large file sizes are being tested, as a way of quickly determining whether an illegal storage access has been made by another initiator.

I/O Behavior – Modify the behavioral aspects of I/O in a test.

Specify Burst Interval – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the burst interval duration.

Specify Thread Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before issuing the next thread. This requires a multi-threaded test definition.

For the start delay: 1) The first thread is issued. 2) There is a pause (for the time value of the start delay.) 3) The next thread is issued. 4) There is a pause (for the time value of the start delay.) 5) The next thread is issued. 6) and so forth.

Specify Target Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before starting an I/O to the next target. This requires multiple targets in the test plan.

Retry Failed I/O – Select the check box and specify the number retries for failed I/Os.

Retry Delay – When **Retry Failed I/O** is selected, edit the time value in Hours, Minutes, and Seconds to set the delay between retries.

I/O Behavior dropdown list

Keep File Handles Open Between I/Os – Select this option to keep the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. This option is ignored for 'sock'.

Close File Handles After Every I/O – Select this option to close the target file descriptor (handle) and re-open after each FOP.

Keep File Handles and Flush Every I/O – Select this option to sync (flush) after each FOP. This makes a request to the operating system to commit all written data to the target device, but it may not bypass the device cache. This option is ignored for 'sock'.

Triggering – Set up the triggers during your test. It instructs the tools to send a write I/O to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. It also generates additional log files that are extremely useful in regards to debug and analysis. The data value to trigger on occurs in the first two words of the data frame. The options associated with this switch are:

Disable Triggering – disables the triggering option.

Write 0xCACACACA 0xCACACACA on Data Corruption
Write 0xCACACACA 0xDEADBEEF on I/O Error – Writes 0xCACACACA 0xCACACACA for data corruption trigger and 0xCACACACA 0xDEADBEEF for I/O error trigger.

Write 0xDEADDEAD 0xDEADDEAD on Data Corruption
Write 0xDEADDEAD 0xDEADBEEF on I/O Error – Writes 0xDEADDEAD 0xDEADDEAD for data corruption trigger, and 0xDEADDEAD 0xDEADBEEF for I/O error trigger.

Select any of the two previous options to continue testing if data corruption or I/O error are detected, but generate a trigger. A write command is sent to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. Additional log files are also generated that are extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer (or phase or write). Because the data frame is consistent with FC or serial storage, but not parallel storage testing, the trigger can be used to catch I/O disruptions on an analyzer. The I/O trigger is sent when a halt or stuck I/O is detected.

Stop Testing Immediately - No Trigger Written – Exits the application immediately and no trigger is written.

Write Default (0xCACACACA) Trigger and Exit – Writes default (0xCACACACA) trigger and exits immediately.

Trigger External Application – Executes external application when triggers are detected. Enter the application in the **Application** text box. Enter the arguments to use when running an external Application in the **Arguments** text box.

This last option can be used to trigger the Xgig Analyzer to start (trigger) or stop capture. For example, to trigger the Analyzer operating in the domain “My Domain (1,1,1) XGIG01001234”, set the application to triggeranalyzer.cmd and enter the arguments as “My Domain(1,1,1)” XGIG01001234 in the **Arguments** text box.

Target Offsets –

Override Default/Test Plan Offsets – Select this check box to override the MLTT default device base offset setting. By default, I/O starts at a 1MB offset on the specified device.

Use Default Offset – uses the default offset.

Use a Shared Offset – allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently.

Specify Start Offset – specifies the starting offset number. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target device.

Error Handlers – allow you to specify error handlers.

To add an error handler:

1. Click **Add**.
2. Select the **Handled Error Value** from the drop-down list.
3. Select the **Label Value as** an **Error**, **Warning**, or **Information**.
4. Select the **Trigger** behavior from the drop-down menu.
5. Select **Specify Trigger Pattern** to enter the trigger pattern.
6. Specify the **Exit Mode** from the drop-down list.
7. Select **Specify Retries** to set number of retries for that error handler.

To remove an error handler:

1. Select the error handler from the **Error Handlers** list.
2. Click **Remove**.

Advanced I/O Tab

The **Advanced I/O** tab allows you to specify custom read/write and I/O mixes.

Advanced Read / Write Mix

When you make changes in this area, the **Read / Write Mix** , the **I/O Operation Size**, and the **Base File Size on I/O Operation Size** settings on the **I/O Payload** tab are rendered void and as such this area is not displayed on the tab.

To specify a custom read/write mix:

1. Select the **Specify Custom Read / Write Mix** check box.
2. Click **Add** to add a new custom read/write mix.
3. Select the newly added custom read/write mix from the list.
4. Click the **Rebalance to 100%** button to automatically change the access percentage values of the custom read/write mixes to total 100%.
5. Use the options in the **Read/Write Specification** panel to customize the read/write mix.

To remove a custom read/write mix:

1. Select the custom read/write mix from the list.
2. Click **Remove**.

Advanced I/O Mix

When you make changes in this area, the **I/O Type** settings on the **I/O Payload** tab are rendered void and as such this area is not displayed on the tab.

To specify a custom I/O mix:

1. Select the **Specify Custom I/O Mix** check box.
2. Set the value for **% Forward Sequential**.
3. Set the value or **% Backward Sequential**.
4. Set the value for **% Random**.

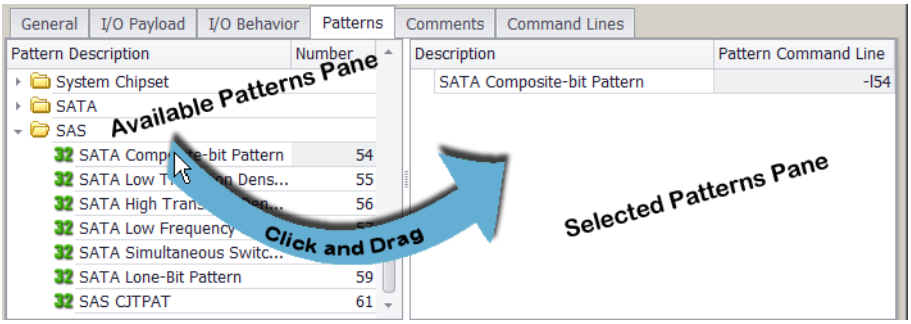
Patterns Tab

The **Patterns** tab allows you to add specific patterns to the test, such as flip/flop patterns, inverted patterns, pattern reversals, data scrambling, or unique data patterns.

Available Patterns – This pane on the upper left of the tab page lists the patterns available for the tests. Several folders are displayed for each available category. Click on the plus/minus sign beside the category type folder to show the list of patterns available in that category.

Selected Patterns – This pane on the upper left of the tab page shows the selected patterns for the current test configuration. This pane displays the test description, the test number, and the command line for the test.

Figure 52: Available Patterns and Selected Patterns



To add a pattern for the current test configuration, click the desired pattern name in the **Available Patterns** pane and drag it into the **Selected Patterns** pane. You can also click a folder and drag it to the **Selected Patterns** pane to add all of the patterns in the folder.

To remove a pattern from the **Selected Patterns** pane, select it and press **Delete** on your keyboard. You can select multiple patterns to delete using the keyboard's Shift or Ctrl buttons.

Pattern Editor Tab

This tab shows the description and the settings for the selected pattern in the **Selected Patterns** pane. The description of the selected pattern is displayed directly beneath the tab name. The options change for the various types of patterns. Select the options for the specific test being performed.

Invert Patterns – This option causes a bit inversion of the data pattern with each transition cycle and is often used to create bit-blink variations over bus architectures.

Use Pattern Reversals – Most data patterns reverse after each FOP (forward, then backward). In some tests (multi-mode, for example), data pattern reversals may look like false data corruptions. Reversals should be allowed anytime data comparisons are being performed as a means of insuring that stale data is not being read.

Reset Pattern Each Cycle – This option causes a "flip/flop" variation to occur within the blinking data pattern. The term "flip/flop" means that the pattern starts at an initial value, inverts (blinks) the value, returns to the initial value, then walks a bit and repeats the sequence.

Scramble Data – Shows options to pre-scramble data patterns according to SAS or SATA specifications. When these patterns are written by MLTT, hardware scrambling will have the effect of de-scrambling the data into the desired pattern. This is an effective means of signal integrity testing on these architectures when combined with the Fibre Channel data patterns. The SAS and SATA options will automatically use default frame data lengths for the scrambler reset. The data length/reset interval can be overridden by specifying the data length in bytes.

No Data Scrambling – No scrambling of data patterns.

SAS Data Scrambling – Pre-scrambles data patterns according to SAS specifications.

SATA Data Scrambling – Pre-scrambles data patterns according to SATA specifications.

Scramble Reset Interval – When you specify a reset interval, you override the data length/reset interval for the scrambled pattern. Select the **Specify Reset Interval** check box and then edit the number to specify the data length. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Data Pattern Cycle Length – The cycle length indicates the number of times to repeat each cycle of a data pattern before moving to the next unit. Select the **Cycle Data Pattern** check box and set the cycle length. In general, the unit of data pattern refers to its length in bytes or bits. An example use of this option is to run an 8-bit pattern four times to produce, effectively, a 32-bit pattern. In this case, each byte is run four times before moving on to the next byte.

Phase shift – This pertains to most blinking data patterns. If the **Use Default Phase Shift** or **Specify Phase Cycle Length** options are selected, the data pattern shifts at the specified cycle length, such that the square wave, created by the on/off bits in the blinking byte values, reverses. The frequency of this shift is determined by the cycle length setting. Cycle length multiplied by pattern length determines the shift frequency.

To use this feature, select either **Use Default Phase Shift** or **Phase Cycle Length**. When **Phase Cycle Length** is selected, enter the number of cycles to run before doing the phase shift. Change this value by entering a value in the text box or using the numeric spinner.

Data Pattern Specification – Allows you to specify a static value to use as the static repeating pattern if you don't want to use with the default. The default pattern for those is to use the thread number for the pattern value. This is repeated continuously. Change this value using the numeric spinner.

Random Seed – Specifies an initialization value for the pseudorandom number generator used to generate a random pattern.

Walking Bit Options – This walks an opposing bit across the sequence when the pattern is a blinking pattern.

Note: Each of the Walking Bit options are shown in Table 6. The opposing bits are shaded in the table so the walking effect can be seen easily.

Table 6: Walking Bits Using an 8-bit Blinking Example

Do Not Use Walking Bits	Walk Bits on 'ON' Cycle	Walk Bits on 'OFF' Cycle	Walk Bits on Both Cycle
00000000	00000000	10000000	10000000
11111111	01111111	11111111	01111111
00000000	00000000	01000000	01000000
11111111	10111111	11111111	10111111
00000000	00000000	00100000	00100000
11111111	11011111	11111111	11011111
00000000	00000000	00010000	00010000
11111111	11101111	11111111	11101111
00000000	00000000	00001000	00001000
11111111	11110111	11111111	11110111
00000000	00000000	00000100	00000100
11111111	11111011	11111111	11111011
00000000	00000000	00000010	00000010
11111111	11111101	11111111	11111101
00000000	00000000	00000001	00000001
11111111	11111110	11111111	11111110

Do Not Use Walking Bits – Walking bits are not used.

Walk Bits on 'ON' Cycle – Walking bits only walks "0" across the "1's" cycle.

Walk Bits on 'OFF' Cycle – Walking bits only walks "1" across the "0's" cycle.

Walk Bits on Both Cycle – Walking bits are walked across both cycles.

Hold Pattern for cycles before walking – Keeps the walking bit in its position for the specified number of cycles before advancing the bit to its the next position. The number of cycles is maintained at each of the bit's walking positions. Change this value by entering a value in the text box or using the numeric spinner.

Set Blink Length – Sets the length of the "ON" ('1') bits. Change this value by entering a value in the text box or using the numeric spinner.

Hexadecimal Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in hexadecimal format.

Binary Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in binary format.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option. This is useful when using any of the modes that create multiple tests with one configuration file (i.e. ranged values, cycle read/write modes, or multiple data patterns, etc.)

Integrity Configuration Editor

The Integrity configuration editor allows you to edit the most common and useful options when you want to ensure that a target is writing and reading data correctly.

To open the editor, double-click the Integrity configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: Some of the options on the editor may be grayed out based on the methodology selected or other option dependencies. The editor opens in the same mode, depending on the mode when it was closed.

The editor has six tabs for specifying testing parameters:

- General (See page 80)
- I/O Payload (See page 81)
- I/O Behavior (See page 84)
- Patterns (See page 86)
- Comments (See page 89)
- Command Lines (See page 90)

The description of each of these tabs and its parameters in the following pages.

General Tab

The **General** tab shows the following settings for this configuration editor.

Startup Options –

Initial Sample Delay – Sets the delay time to wait before starting the testing. This delay allows devices the allotted time to get setup. The **H** (Hour), **M** (Minute), and **S** (Second) numeric spinners allow you to delay the start of the testing up to 23 hours, 59 minutes, and 59 seconds.

Throughput Limitation – sets the limitations for the maximum I/O throughput

Infinity – sets the maximum I/O throughput to have no limitations.

Every Target – sets the **Max I/O Throughput** value to apply to every target.

Every Thread – sets the **Max I/O Throughput** value to apply to every thread.

Max I/O Throughput – provides the maximum I/O throughput limitation value that is applied to every target or thread.

External Application – runs any application of your choosing after every test has been run. For example, if you want to run an independent application to collect data (such as a log file) from a device, you can use this to start the application.

The **Run External Application After Test** check box must be selected before you can select the application and wait times.

Application allows for you to browse to and select the desired application.

Wait for External Application allows you to input a period to wait for the application to start to run. Tests tools will continue testing once the external application begins running.

I/O Payload Tab

The **I/O Payload** tab shows the basic parameters for a Test Tool test, such as the testing style (synchronous or asynchronous), testing threads, queue depth, I/O operation size, I/O type, I/O marking and signing, and logging level.

Performance – Select the **Enable Performance mode** check box to enable the performance mode. This mode increases the speed of testing by optimizing the use of internal memory buffers. When this check box is selected, the following settings are automatically set:

Table 7: Performance Mode Settings

GUI Setting	Command Line Setting
I/O Marking and Signing is set to **No I/O Markings** No I/O signatures are applied to each sector of every write.	`-u` (page 177)
Data Compare Mode is set to **Disable Data Comparisons** Data comparisons are turned off.	`-n` (page 177)
I/O Behavior is set to **Keep File Handles Open Between I/Os** Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP.	`-o` (page 166)
Use Pattern Reversals check box is not selected (cleared) Leaves the data patterns reversal after each FOP (forward, then backward) turned off.	`-N` (page 174)

Testing Style – Select **Synchronous (Pain)** or **Asynchronous (Maim)**.

When **Synchronous (Pain)** is selected, set the **Testing Threads** area. Also select the appropriate SCSI passthrough option from the drop-down menu. The options are listed below:

- SCSI Passthrough Off
- READ/WRITE 10
- READ/WRITE 10 + FUA (Forced Unit Access)
- READ/WRITE 16
- READ/WRITE 16 + FUA (Forced Unit Access)

When **Asynchronous (Maim)** is selected, set the **Testing Threads** area. Also set the **Queue Depth** area.

Testing Threads – Set the threads for testing. Each thread executes a single I/O at a time, with each thread starting at a different base offset. The number of threads successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

You can also set a range of threads to test by selecting the **Test a Range of Threads** check box. Set the **Thread Count Start** and the **Thread Count End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the tested threads through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Queue Depth – (displayed for Asynchronous (Maim) only) Set the queue depth.

The queue depth is the maximum number of current I/Os to execute in a single worker thread. The value of queue depth successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the queue depth by entering the value or clicking the **Queue Depth** numeric spinner.

You can also set a range for the queue depth by selecting the **Test a Range of Queue Depths** check box. Set the **Queue Depth Start** and the **Queue Depth End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

The **Queue Depth** has two additional options, **Keep Queue Depth Static** and **Strict Sequential**.

Select **Keep Queue Depth Static** to add `-m16` option to the command line to use continuous queuing. If this option is not selected, by default burst queuing will be used.

Select **Strict Sequential** to add the -m1 option to the command line which will make the test have continuous queuing and strict sequential access.

Testing Sizes – Select the **I/O Operation Size** to be used for testing.

You can set the I/O operation size by entering the value or clicking the **I/O Operation Size** numeric spinner. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Edit the number or click the numeric spinner. Select the unit by clicking the drop-down button.

You can also set a range for the I/O operation size by selecting the **Test a Range of I/O Operation Sizes** check box. Set the **I/O Operation Size Start** and the **I/O Operation Size End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Additional **Testing Sizes** settings include:

Base File Size on I/O Operation Size – Select this option to use the **I/O Operation Size** as basis for the testing area. For synchronous tests (pain), the file size is equal to the block size. For asynchronous tests (maim), the file size is the block size multiplied by the queue depth.

Specify Testing Area – Select this button and specify the file size or disk area to use per worker thread. The size can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Test Using the Entire Target – Select this check box to use the entire target for the test. This is not applicable for file system and memory testing.

I/O Type – Choose from **Forwards Only**, **Alternate Between Forwards and Backwards**, **First FOP Forwards, Rest Backwards**, **Backwards Only**, and **Custom Mixture**.

When you select **Custom Mixture**, use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the percentage of randomness, while sliding it to the right increases the sequential operation.

This slider works in unison with the **% Random** and the **% Sequential** boxes so that the sum of both values is 100 percent. As the slider is moved, the values in the **% Random** and the **% Sequential** box values change to reflect the slider position.

Likewise, when either the **% Random** or the **% Sequential** boxes are changed by entering a value or using the numeric spinner, the other box and the slider are adjusted to reflect the change.

I/O Marking and Signing – Select the I/O Marking and Signing option from the drop-down list. For details on I/O signatures, refer to Appendix D, “I/O Signatures”.

No I/O Markings – No I/O signatures are applied to each sector of every write.

Uniquely Mark I/O (Default) – I/O signatures are applied to each sector of every write, unless disabled.

Add Time Stamps to I/O – Select this option to add timestamp for the I/O event as part of the I/O signature. Timestamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Add Time Stamps in Milliseconds to I/O – Select to choose a time stamp for the test session. Select this option to add timestamp (in milliseconds) for the I/O event as part of the I/O signature. Time stamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Override Session Id – Select this to override the default session ID and specify your own. The default session ID is a semi-unique field in the I/O signature that created using the hex values of the last two characters of the target name. It is way to help identify the host and target when you are debugging.

Logging Level – Select the type of logging you want for the configuration to indicate the level of information to be posted to the log file. The default option posts the maximum amount of information which is helpful for analyzing errors.

Standard logfile generation/output – Default option, includes detailed headers and console performance output.

Outputs to logfile in test performance format (minimal logging) – Removes headers and logs performance output only.

No outputs to logfile, minimal screen outputs, PRF log summary – No .log file generated, and logs minimal screen output.

Disable CSV log – Standard logging, but .csv file will not be created.

Single line output with system name, performance, and errors – Includes system name and other details on single output lines for easier importation or parsing.

Disable completion statistics in PRF file – Disables completion calculations and output. In IOPS intensive tests where the CPU is heavily taxed, using this option may result in a slight performance gain.

Enable logging of informational events in Windows event log – This option will send informational output to Windows event log, such as test start and stop details.

Command Line – As options are selected, the equivalent command line settings appear in the textbox. See Chapter 4, “Using the Command Line Switches’’ for details of the command line settings.

I/O Behavior Tab

The **I/O Behavior** tab allows you to specify data comparisons, set I/O behavior, setup triggering options based on test results, use non-default target offsets, and setup error handlers.

Data Compare Mode – Click the drop-down list to choose a data comparison mode. A byte-for-byte data comparison of write and read data will catch any possible data corruption. Data comparison is usually recommended except in special cases, such as when the overhead of full buffer comparisons decreases the I/O throughput to the target.

Disable Data Comparisons – Data comparisons are turned off.

Full byte-for-byte Comparison – Each byte is checked for integrity (default value).

Signature Comparison Only – (2-3 words every 512 bytes) Checks only the unique I/O signature in the data buffer. This substantially reduces processor utilization in the host system.

Session ID Comparison Only – (16 bit ID at 2nd word every 512 bytes) Compares only the session ID used in the data signature. The session ID is generated from the host and target names. This option can be used with the “Override default session ID” option and is usually used in multi-initiator setups as a quick way of verifying that an initiator’s storage has not been written to by another initiator.

Session ID Comparison, Followed by Full – This option is a combination of the **Session ID Comparison Only** check with a full data comparison check. This is another method used in multi-initiator setups, typically used when large file sizes are being tested, as a way of quickly determining whether an illegal storage access has been made by another initiator.

I/O Behavior – Modify the behavioral aspects of I/O in a test.

Specify Burst Interval – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the burst interval duration.

Specify Thread Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before issuing the next thread. This requires a multi-threaded test definition.

For the start delay: 1) The first thread is issued. 2) There is a pause (for the time value of the start delay.) 3) The next thread is issued. 4) There is a pause (for the time value of the start delay.) 5) The next thread is issued. 6) and so forth.

Specify Target Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before starting an I/O to the next target. This requires multiple targets in the test plan.

Retry Failed I/O – Select the check box and specify the number retries for failed I/Os.

Retry Delay – When **Retry Failed I/O** is selected, edit the time value in Hours, Minutes, and Seconds to set the delay between retries.

I/O Behavior dropdown list

Keep File Handles Open Between I/Os – Select this option to keep the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. This option is ignored for 'sock'.

Close File Handles After Every I/O – Select this option to close the target file descriptor (handle) and re-open after each FOP.

Keep File Handles and Flush Every I/O – Select this option to sync (flush) after each FOP. This makes a request to the operating system to commit all written data to the target device, but it may not bypass the device cache. This option is ignored for 'sock'.

Triggering – Set up the triggers during your test. It instructs the tools to send a write I/O to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. It also generates additional log files that are extremely useful in regards to debug and analysis. The data value to trigger on occurs in the first two words of the data frame. The options associated with this switch are:

Disable Triggering – disables the triggering option.

Write 0xCACACACA 0xCACACACA on Data Corruption,
Write 0xCACACACA 0xDEADBEEF on I/O Error – Writes 0xCACACACA 0xCACACACA for data corruption trigger and 0xCACACACA 0xDEADBEEF for I/O error trigger.

Writes 0xDEADDEAD 0xDEADDEAD on Data Corruption,
Write 0xDEADDEAD 0xDEADBEEF on I/O Error – Writes 0xDEADDEAD 0xDEADDEAD for data corruption trigger, and 0xDEADDEAD 0xDEADBEEF for I/O error trigger.

Select any of the two previous options to continue testing if data corruption or I/O error are detected, but generate a trigger. A write command is sent to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. Additional log files are also generated that are extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer (or phase or write). Because the data frame is consistent with FC or serial storage, but not parallel storage testing, the trigger can be used to catch I/O disruptions on an analyzer. The I/O trigger is sent when a halt or stuck I/O is detected.

Stop Testing Immediately - No Trigger Written – Exits the application immediately and no trigger is written.

Write Default (0xCACACACA) Trigger and Exit – Writes default (0xCACACACA) trigger and exits immediately.

Trigger External Application – Executes external application when triggers are detected. Enter the application in the **Application** text box. Enter the arguments to use when running an external Application in the **Arguments** text box.

This last option can be used to trigger the Xgig Analyzer to start (trigger) or stop capture. For example, to trigger the Analyzer operating in the domain “My Domain (1,1,1) XGIG01001234”, set the application to triggeranalyzer.cmd and enter the arguments as “My Domain(1,1,1)” XGIG01001234 in the **Arguments** text box.

Target Offsets –

Override Default/Test Plan Offsets – Select this check box to override the MLTT default device base offset setting. By default, I/O starts at a 1MB offset on the specified device.

Use Default Offset – uses the default offset.

Use a Shared Offset – allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently.

Specify Start Offset – specifies the starting offset number. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target device.

Error Handlers – allows you to specify error handlers.

To add an error handler:

1. Click **Add**.
2. Select the **Handled Error Value** from the drop-down list.
3. Select the **Label Value as** an **Error**, **Warning**, or **Information**.
4. Select the **Trigger** behavior from the drop-down menu.
5. Select **Specify Trigger Pattern** to enter the trigger pattern.
6. Specify the **Exit Mode** from the drop-down list.
7. Select **Specify Retries** so that you can set number of retries for that error handler.

To remove an error handler:

1. Select the error handler from the **Error Handlers** list.
2. Click **Remove**.

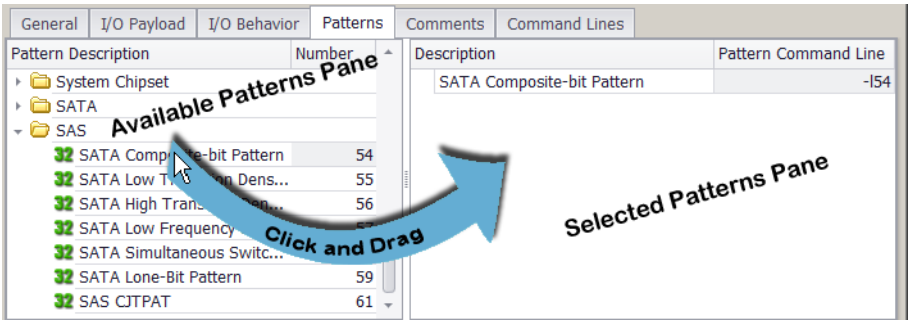
Patterns Tab

The **Patterns** tab allows you to add specific patterns to the test, such as flip/flop patterns, inverted patterns, pattern reversals, data scrambling, or unique data patterns.

Available Patterns – This pane on the upper left of the tab page lists the patterns available for the tests. Several folders are displayed for each available category. Click on the plus/minus sign beside the category type folder to show the list of patterns available in that category.

Selected Patterns – This pane on the upper left of the tab page shows the selected patterns for the current test configuration. This pane displays the test description, the test number, and the command line for the test.

Figure 53: Available Patterns and Selected Patterns



To add a pattern for the current test configuration, click the desired pattern name in the **Available Patterns** pane and drag it into the **Selected Patterns** pane. You can also click a folder and drag it to the **Selected Patterns** pane to add all of the patterns in the folder.

To remove a pattern from the **Selected Patterns** pane, select it and press **Delete** on your keyboard. You can select multiple patterns to delete using the keyboard's Shift or Ctrl buttons.

Pattern Editor Tab

This tab shows the description and the settings for the selected pattern in the **Selected Patterns** pane. The description of the selected pattern is displayed directly beneath the tab name. The options change for the various types of patterns. Select the options for the specific test being performed.

Invert Patterns – This option causes a bit inversion of the data pattern with each transition cycle and is often used to create bit-blink variations over bus architectures.

Use Pattern Reversals – Most data patterns reverse after each FOP (forward, then backward). In some tests (multi-mode, for example), data pattern reversals may look like false data corruptions. Reversals should be allowed anytime data comparisons are being performed as a means of insuring that stale data is not being read.

Reset Pattern Each Cycle – This option causes a "flip/flop" variation to occur within the blinking data pattern. The term "flip/flop" means that the pattern starts at an initial value, inverts (blinks) the value, returns to the initial value, then walks a bit and repeats the sequence.

Scramble Data – Shows options to pre-scramble data patterns according to SAS or SATA specifications. When these patterns are written by MLTT, hardware scrambling will have the effect of de-scrambling the data into the desired pattern. This is an effective means of signal integrity testing on these architectures when combined with the Fibre Channel data patterns. The SAS and SATA options will automatically use default frame data lengths for the scrambler reset. The data length/reset interval can be overridden by specifying the data length in bytes.

No Data Scrambling – No scrambling of data patterns.

SAS Data Scrambling – Pre-scrambles data patterns according to SAS specifications.

SATA Data Scrambling – Pre-scrambles data patterns according to SATA specifications.

Scramble Reset Interval – When you specify a reset interval, you override the data length/reset interval for the scrambled pattern. Select the **Specify Reset Interval** check box and then edit the number to specify the data length. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Data Pattern Cycle Length – The cycle length indicates the number of times to repeat each cycle of a data pattern before moving to the next unit. Select the **Cycle Data Pattern** check box and set the cycle length. In general, the unit of data pattern refers to its length in bytes or bits. An example use of this option is to run an 8-bit pattern four times to produce, effectively, a 32-bit pattern. In this case, each byte is run four times before moving on to the next byte.

Phase shift – This pertains to most blinking data patterns. If the **Use Default Phase Shift** or **Specify Phase Cycle Length** options are selected, the data pattern shifts at the specified cycle length, such that the square wave, created by the on/off bits in the blinking byte values, reverses. The frequency of this shift is determined by the cycle length setting. Cycle length multiplied by pattern length determines the shift frequency.

To use this feature, select either **Use Default Phase Shift** or **Phase Cycle Length**. When **Phase Cycle Length** is selected, enter the number of cycles to run before doing the phase shift. Change this value by entering a value in the text box or using the numeric spinner.

Data Pattern Specification – Allows you to specify a static value to use as the static repeating pattern if you don't want to use with the default. The default pattern for those is to use the thread number for the pattern value. This is repeated continuously. Change this value using the numeric spinner.

Random Seed – Specifies an initialization value for the pseudorandom number generator used to generate a random pattern.

Walking Bit Options – This walks an opposing bit across the sequence when the pattern is a blinking pattern.

Note: Each of the Walking Bit options are shown in Table 8. The opposing bits are shaded in the table so the walking effect can be seen easily.

Table 8: Walking Bits Using an 8-bit Blinking Example

Do Not Use Walking Bits	Walk Bits on 'ON' Cycle	Walk Bits on 'OFF' Cycle	Walk Bits on Both Cycle
00000000	00000000	10000000	10000000
11111111	01111111	11111111	01111111
00000000	00000000	01000000	01000000
11111111	10111111	11111111	10111111
00000000	00000000	00100000	00100000
11111111	11011111	11111111	11011111
00000000	00000000	00010000	00010000
11111111	11101111	11111111	11101111
00000000	00000000	00001000	00001000
11111111	11110111	11111111	11110111
00000000	00000000	00000100	00000100
11111111	11111011	11111111	11111011
00000000	00000000	00000010	00000010
11111111	11111101	11111111	11111101
00000000	00000000	00000001	00000001
11111111	11111110	11111111	11111110

Do Not Use Walking Bits – Walking bits are not used.

Walk Bits on 'ON' Cycle – Walking bits only walks "0" across the "1's" cycle.

Walk Bits on 'OFF' Cycle – Walking bits only walks "1" across the "0's" cycle.

Walk Bits on Both Cycle – Walking bits are walked across both cycles.

Hold Pattern for cycles before walking – Keeps the walking bit in its position for the specified number of cycles before advancing the bit to its the next position. The number of cycles is maintained at each of the bit's walking positions. Change this value by entering a value in the text box or using the numeric spinner.

Set Blink Length – Sets the length of the "ON" ('1') bits. Change this value by entering a value in the text box or using the numeric spinner.

Hexadecimal Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in hexadecimal format.

Binary Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in binary format.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option. This is useful when using any of the modes that create multiple tests with one configuration file (i.e. ranged values, cycle read/write modes, or multiple data patterns, etc.)

Performance Configuration Editor

The Performance configuration editor allows you to make a configuration that ensures a device is performing at the expected speed. It is intended for users that find they are not accomplishing their testing requirements through Integrity configurations provided in the default samples.

To open the editor, double-click the Performance configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: Some of the options on the editor may be grayed out based on the methodology selected or other option dependencies. The editor opens in the same mode depending on the mode when it was closed.

The editor has four tabs for specifying testing parameters:

- General (See page 91)
- I/O Payload (See page 92)
- Comments (See page 96)
- Command Lines (See page 96)

The description of each of these tabs and its parameters in the following pages.

General Tab

The **General** tab shows the following settings for this configuration editor.

Startup Options –

Initial Sample Delay – Sets the delay time to wait before starting the testing. This delay allows devices the allotted time to get setup. The **H** (Hour), **M** (Minute), and **S** (Second) numeric spinners allow you to delay the start of the testing up to 23 hours, 59 minutes, and 59 seconds.

Throughput Limitation – sets the limitations for the maximum I/O throughput

Infinity – sets the maximum I/O throughput to have no limitations.

Every Target – sets the **Max I/O Throughput** value to apply to every target.

Every Thread – sets the **Max I/O Throughput** value to apply to every thread.

Max I/O Throughput – provides the maximum I/O throughput limitation value that is applied to every target or thread.

External Application – runs any application of your choosing after every test has been run. For example, if you want to run an independent application to collect data (such as a log file) from a device, you can use this to start the application.

The **Run External Application After Test** check box must be selected before you can select the application and wait times.

Application allows for you to browse to and select the desired application.

Wait for External Application allows you to input a period to wait for the application to start to run. Tests tools will continue testing once the external application begins running.

Steady State

Steady State determines the steady state for a target across five consecutive test runs. With the test plan set to run indefinitely or several times (5 times or more), when steady state is achieved, the test plan will be stopped when the test plan iteration is complete. The planning group will start the next test plan.

If steady state is not achieved during the specified number of test runs, the test plan will complete its last iteration and testing is terminated. Subsequent test plans in the planning group are ignored. If you select to run the test plan indefinitely and steady state is not achieved, you will need to stop the test manually.

Select the **Check for Steady State** check box to enable this feature. Once the check box is selected, you can set the following options:

Tag text box allows you to enter an arbitrary string that is not 'r', 'iops', 'mbps' or 'lat' which can be used to uniquely identify a steady state testing case. This tag will be added to the steady-state.csv file name.

Tracking Variable group allows you to select one of the Tracking radio buttons and set the Deviation percentages.

Tracking radio buttons:

IOPS Tracks IOPS for steady state. (default value)

MBPS Tracks MBPS for steady state.

IO Latency Tracks I/O latency for steady state.

Deviation percentages:

% Range Deviation: Allowed deviation of minimum and maximum tracked values from the average. The default value is 20%.

% Slope Deviation: Allowed deviation of minimum and maximum points in a best linear fit line through the tracked values. The default value is 10%.

I/O Payload Tab

The **I/O Payload** tab shows the basic parameters for a Test Tool test, such as the testing style (synchronous or asynchronous), testing threads, queue depth, testing sizes, I/O operation sizes, read/write mix, I/O type, and logging levels.

Performance – Select the **Enable Performance mode** check box to enable the performance mode. This mode increases the speed of testing by optimizing the use of internal memory buffers. When this check box is selected, the following settings are automatically set:

Table 9: Performance Mode Settings

GUI Setting	Command Line Setting
I/O Marking and Signing is set to **No I/O Markings** No I/O signatures are applied to each sector of every write.	`-u` (page 177)
Data Compare Mode is set to **Disable Data Comparisons** Data comparisons are turned off.	`-n` (page 177)
I/O Behavior is set to **Keep File Handles Open Between I/Os** Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP.	`-o` (page 166)
Use Pattern Reversals check box is not selected (cleared) Leaves the data patterns reversal after each FOP (forward, then backward) turned off.	`-N` (page 174)

Testing Style – Select **Synchronous (Pain)** or **Asynchronous (Maim)**.

When **Synchronous (Pain)** is selected, set the **Testing Threads** area. Also select the appropriate SCSI passthrough option from the drop-down menu. The options are listed below:

- SCSI Passthrough Off
- READ/WRITE 10
- READ/WRITE 10 + FUA (Forced Unit Access)
- READ/WRITE 16
- READ/WRITE 16 + FUA (Forced Unit Access)

When **Asynchronous (Maim)** is selected, set the **Testing Threads** area. Also set the **Queue Depth** area.

Testing Threads – Set the threads for testing. Each thread executes a single I/O at a time, with each thread starting at a different base offset. The number of threads successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

You can also set a range of threads to test by selecting the **Test a Range of Threads** check box. Set the **Thread Count Start** and the **Thread Count End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the tested threads through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Queue Depth – (displayed for Asynchronous (Maim) only) Set the queue depth.

The queue depth is the maximum number of current I/Os to execute in a single worker thread. The value of queue depth successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the queue depth by entering the value or clicking the **Queue Depth** numeric spinner.

You can also set a range for the queue depth by selecting the **Test a Range of Queue Depths** check box. Set the **Queue Depth Start** and the **Queue Depth End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

The **Queue Depth** has two additional options, **Keep Queue Depth Static** and **Strict Sequential**.

Select **Keep Queue Depth Static** to add `-m16` option to the command line to use continuous queuing. If this option is not selected, by default burst queuing will be used.

Select **Strict Sequential** to add the -m1 option to the command line which will make the test have continuous queuing and strict sequential access.

Testing Sizes – Select the **I/O Operation Size** to be used for testing.

You can set the I/O operation size by entering the value or clicking the **I/O Operation Size** numeric spinner. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Edit the number or click the numeric spinner. Select the unit by clicking the drop-down button.

You can also set a range for the I/O operation size by selecting the **Test a Range of I/O Operation Sizes** check box. Set the **I/O Operation Size Start** and the **I/O Operation Size End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Additional **Testing Sizes** settings include:

Base File Size on I/O Operation Size – Select this option to use the **I/O Operation Size** as basis for the testing area. For synchronous tests (pain), the file size is equal to the block size. For asynchronous tests (maim), the file size is the block size multiplied by the queue depth.

Specify Testing Area – Select this button and specify the file size or disk area to use per worker thread. The size can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Test Using the Entire Target – Select this check box to use the entire target for the test. This is not applicable for file system and memory testing.

Read/Write Mix – Select the read or write mode of the test.

Cycle Through the Read / Write Modes – Select this check box to cycle through the various read/write modes. When this check box is selected, the following Read/Write mix settings are not applicable and they are not displayed.

Read / Write – Select this option to have a balance of read and write testing.

Read Only – Select this option to have a read-only test. If this option is selected, the **Force Initial Write** and **Do Not Perform Initial Write** radio buttons and will be available.

Force Initial Write – Performs a Write I/O once followed by continuous Reads.

Do Not Perform Initial Write – Performs pure read-only I/Os. This also disables data comparison automatically.

Write Only – Select this option to have a write-only test.

Specify Custom Read/Write Mix – Setting the slider in the middle gives 50%/50% chance to Read/Write to be the next IO. This results in a random mix of reads/writes.

Use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the write operations, while sliding it to the right increases the read operations.

- Sliding to the left most makes it a **Write only** test, where it writes a test pattern but does not read it.
- Sliding to the right most makes it a **Read only** test, where it only returns the data that exists in the file or device area.
- Setting the slider to the middle makes it a **Read/Write** test, where it repeatedly writes a test pattern and then reads it back.

I/O Type – Choose from **Forwards Only**, **Alternate Between Forwards and Backwards**, **First FOP Forwards, Rest Backwards**, **Backwards Only**, and **Custom Mixture**.

When you select **Custom Mixture**, use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the percentage of randomness, while sliding it to the right increases the sequential operation.

This slider works in unison with the **% Random** and the **% Sequential** boxes so that the sum of both values is 100 percent. As the slider is moved, the values in the **% Random** and the **% Sequential** box values change to reflect the slider position.

Likewise, when either the **% Random** or the **% Sequential** boxes are changed by entering a value or using the numeric spinner, the other box and the slider are adjusted to reflect the change.

I/O Marking and Signing – Select the I/O Marking and Signing option from the drop-down list. For details on I/O signatures, refer to Appendix D, “I/O Signatures”.

No I/O Markings – No I/O signatures are applied to each sector of every write.

Uniquely Mark I/O (Default) – I/O signatures are applied to each sector of every write, unless disabled.

Add Time Stamps to I/O – Select this option to add timestamp for the I/O event as part of the I/O signature. Timestamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Add Time Stamps in Milliseconds – Select to choose a time stamp for the test session. Select this option to add timestamp (in milliseconds) for the I/O event as part of the I/O signature. Time stamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Override Session Id – Select this to override the default session ID and specify your own. The default session ID is a semi-unique field in the I/O signature that created using the hex values of the last two characters of the target name. It is way to help identify the host and target when you are debugging.

Logging Level – Select the type of logging you want for the configuration to indicate the level of information to be posted to the log file. The default option posts the maximum amount of information which is helpful for analyzing errors.

Standard logfile generation/output – Default option, includes detailed headers and console performance output.

Outputs to logfile in test performance format (minimal logging) – Removes headers and logs performance output only.

No outputs to logfile, minimal screen outputs, PRF log summary – No .log file generated, and logs minimal screen output.

Disable CSV log – Standard logging, but .csv file will not be created.

Single line output with system name, performance, and errors – Includes system name and other details on single output lines for easier importation or parsing.

Disable completion statistics in PRF file – Disables completion calculations and output. In IOPS intensive tests where the CPU is heavily taxed, using this option may result in a slight performance gain.

Enable logging of informational events in Windows event log – This option will send informational output to Windows event log, such as test start and stop details.

Command Line – As options are selected, the equivalent command line settings appear in the textbox. See Chapter 4, "Using the Command Line Switches'' for details of the command line settings.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option. This is useful when using any of the modes that create multiple tests with one configuration file (i.e. ranged values, cycle read/write modes, or multiple data patterns, etc.)

Storage CLI Configuration Editor

The Storage CLI configuration editor saves pain and maim command lines so the command lines can be sent from the configuration editor.

To open the editor, double-click the Storage CLI configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: The pain and maim commands sent from this editor do not have error checking. If invalid commands are entered and sent, these commands are ignored and are not reported.

The editor has two tabs for specifying testing parameters and information:

- Command Line
- Comments

Command Line Tab

The **Command Line** tab allows you to enter and send pain and maim commands. This is useful when you want to run a command from the GUI.

The **Paste** button pastes a previously-copied command line to the configuration editor.

The **Copy** button copies the configuration editor's command line once it is selected.

For sock commands, refer to "Network CLI Configuration Editor" on page 118.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Socket Configuration Editor

The Socket configuration editor allows you to edit available options relevant to network tests with TCP/IP sockets.

To open the editor, double-click the Socket configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

To use more advanced parameters, click the **Show Ranged Controls** check box.

The editor has seven tabs for specifying testing parameters:

- General (See page 99)
- I/O Payload (See page 99)
- I/O Behavior (See page 102)
- Advanced I/O (See page 104)
- Patterns (See page 105)
- Comments (See page 108)
- Command Lines (See page 108)

The description of each of these tabs and its parameters in the following pages.

General Tab

The **General** tab shows the following settings for this configuration editor.

Startup Options –

Initial Sample Delay – Sets the delay time to wait before starting the testing. This delay allows devices the allotted time to get setup. The **H** (Hour), **M** (Minute), and **S** (Second) numeric spinners allow you to delay the start of the testing up to 23 hours, 59 minutes, and 59 seconds.

Throughput Limitation – sets the limitations for the maximum I/O throughput

Infinity – sets the maximum I/O throughput to have no limitations.

Every Target – sets the **Max I/O Throughput** value to apply to every target.

Every Thread – sets the **Max I/O Throughput** value to apply to every thread.

Max I/O Throughput – provides the maximum I/O throughput limitation value that is applied to every target or thread.

External Application – runs any application of your choosing after every test has been run. For example, if you want to run an independent application to collect data (such as a log file) from a device, you can use this to start the application.

The **Run External Application After Test** check box must be selected before you can select the application and wait times.

Application allows for you to browse to and select the desired application.

Wait for External Application allows you to input a period to wait for the application to start to run. Tests tools will continue testing once the external application begins running.

Socket Type – Sets the protocol for the Socket configuration.

TCP (Transmission Control Protocol) – Sets the socket configuration protocol to TCP.

UDP (User Datagram Protocol) – Sets the socket configuration protocol to UDP.

I/O Payload Tab

The **I/O Payload** tab shows the basic parameters for a Test Tool test, such as the testing threads, I/O operation size, read/write mix, I/O marking and signing, and logging level. Queue depth is also available after other parameters on the tab are changed.

Performance – Select the **Enable Performance mode** check box to enable the performance mode. This mode increases the speed of testing by optimizing the use of internal memory buffers. When this check box is selected, the following settings are automatically set:

Table 10: Performance Mode Settings

GUI Setting	Command Line Setting
I/O Marking and Signing is set to **No I/O Markings** No I/O signatures are applied to each sector of every write.	`-u` (page 177)
Data Compare Mode is set to **Disable Data Comparisons** Data comparisons are turned off.	`-n` (page 177)
I/O Behavior is set to **Keep File Handles Open Between I/Os** Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP.	`-o` (page 166)
Use Pattern Reversals check box is not selected (cleared) Leaves the data patterns reversal after each FOP (forward, then backward) turned off.	`-N` (page 174)

Testing Threads – Set the threads for testing. Each thread executes a single I/O at a time, with each thread starting at a different base offset. The number of threads successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

You can also set a range of threads to test by selecting the **Test a Range of Threads** check box. Set the **Thread Count Start** and the **Thread Count End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the tested threads through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Queue Depth – (available after other tab parameters are changed) Set the queue depth.

The queue depth is the maximum number of current I/Os to execute in a single worker thread. The value of queue depth successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the queue depth by entering the value or clicking the **Queue Depth** numeric spinner.

You can also set a range for the queue depth by selecting the **Test a Range of Queue Depths** check box. Set the **Queue Depth Start** and the **Queue Depth End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

The **Queue Depth** has two additional options, **Keep Queue Depth Static** and **Strict Sequential**.

Select **Keep Queue Depth Static** to add `-m16` option to the command line to use continuous queuing. If this option is not selected, by default burst queuing will be used.

Select **Strict Sequential** to add the -m1 option to the command line which will make the test have continuous queuing and strict sequential access.

Testing Sizes – Select the **I/O Operation Size** to be used for testing.

You can set the I/O operation size by entering the value or clicking the **I/O Operation Size** numeric spinner. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Edit the number or click the numeric spinner. Select the unit by clicking the drop-down button.

You can also set a range for the I/O operation size by selecting the **Test a Range of I/O Operation Sizes** check box. Set the **I/O Operation Size Start** and the **I/O Operation Size End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

Additional **Testing Sizes** settings include:

Base File Size on I/O Operation Size – Select this option to use the **I/O Operation Size** as basis for the testing area. For synchronous tests (pain), the file size is equal to the block size. For asynchronous tests (maim), the file size is the block size multiplied by the queue depth.

Specify Testing Area – Select this button and specify the file size or disk area to use per worker thread. The size can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Read/Write Mix – Select the read or write mode of the test.

Cycle Through the Read / Write Modes – Select this check box to cycle through the various read/write modes. When this check box is selected, the following Read/Write mix settings are not applicable and they are not displayed.

Read / Write – Select this option to have a balance of read and write testing.

Read Only – Select this option to have a read-only test. If this option is selected, the **Force Initial Write** and **Do Not Perform Initial Write** radio buttons and will be available.

Force Initial Write – Performs a Write I/O once followed by continuous Reads.

Do Not Perform Initial Write – Performs pure read-only I/Os. This also disables data comparison automatically.

Write Only – Select this option to have a write-only test.

Specify Custom Read/Write Mix – Setting the slider in the middle gives 50%/50% chance to Read/Write to be the next IO. This results in a random mix of reads/writes.

Use the slider to set the percentage mix of random and sequential I/Os. Sliding it to the left increases the write operations, while sliding it to the right increases the read operations.

- Sliding to the left most makes it a **Write only** test, where it writes a test pattern but does not read it.
- Sliding to the right most makes it a **Read only** test, where it only returns the data that exists in the file or device area.
- Setting the slider to the middle makes it a **Read/Write** test, where it repeatedly writes a test pattern and then reads it back.

I/O Marking and Signing – Select the I/O Marking and Signing option from the drop-down list. For details on I/O signatures, refer to Appendix D, “I/O Signatures”.

No I/O Markings – No I/O signatures are applied to each sector of every write.

Uniquely Mark I/O (Default) – I/O signatures are applied to each sector of every write, unless disabled.

Add Time Stamps to I/O – Select this option to add timestamp for the I/O event as part of the I/O signature. Timestamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Add Time Stamps in Milliseconds to I/O – Select to choose a time stamp for the test session. Select this option to add timestamp (in milliseconds) for the I/O event as part of the I/O signature. Time stamps are vital components for data integrity checking, and they are also useful for debugging purposes.

Override Session Id - Select this to override the default session ID and specify your own. The default session ID is a semi-unique field in the I/O signature that created using the hex values of the last two characters of the target name. It is way to help identify the host and target when you are debugging.

Logging Level – Select the type of logging you want for the configuration to indicate the level of information to be posted to the log file. The default option posts the maximum amount of information which is helpful for analyzing errors.

Standard logfile generation/output – Default option, includes detailed headers and console performance output.

Outputs to logfile in test performance format (minimal logging) – Removes headers and logs performance output only.

No outputs to logfile, minimal screen outputs, PRF log summary – No .log file generated, and logs minimal screen output.

Disable CSV log – Standard logging, but .csv file will not be created.

Single line output with system name, performance, and errors – Includes system name and other details on single output lines for easier importation or parsing.

Disable completion statistics in PRF file – Disables completion calculations and output. In IOPS intensive tests where the CPU is heavily taxed, using this option may result in a slight performance gain.

Enable logging of informational events in Windows event log – This option will send informational output to Windows event log, such as test start and stop details.

Command Line – As options are selected, the equivalent command line settings appear in the textbox. See Chapter 4, "Using the Command Line Switches'' for details of the command line settings.

I/O Behavior Tab

The **I/O Behavior** tab allows you to specify data comparisons, set I/O behavior, setup triggering options based on test results, use non-default target offsets, and setup error handlers.

Data Compare Mode – Click the drop-down list to choose a data comparison mode. A byte-for-byte data comparison of write and read data will catch any possible data corruption. Data comparison is usually recommended except in special cases, such as when the overhead of full buffer comparisons decreases the I/O throughput to the target.

Disable Data Comparisons – Data comparisons are turned off.

Full byte-for-byte Comparison – Each byte is checked for integrity (default value).

Signature Comparison Only – (2-3 words every 512 bytes) Checks only the unique I/O signature in the data buffer. This substantially reduces processor utilization in the host system.

Session ID Comparison Only – (16 bit ID at 2nd word every 512 bytes) Compares only the session ID used in the data signature. The session ID is generated from the host and target names. This option can be used with the "Override default session ID" option and is usually used in multi-initiator setups as a quick way of verifying that an initiator's storage has not been written to by another initiator.

Session ID Comparison, Followed by Full – This option is a combination of the **Session ID Comparison Only** check with a full data comparison check. This is another method used in multi-initiator setups, typically used when large file sizes are being tested, as a way of quickly determining whether an illegal storage access has been made by another initiator.

I/O Behavior – Modify the behavioral aspects of I/O in a test.

Specify Burst Interval – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the burst interval duration.

Specify Thread Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before issuing the next thread. This requires a multi-threaded test definition.

For the start delay: 1) The first thread is issued. 2) There is a pause (for the time value of the start delay.) 3) The next thread is issued. 4) There is a pause (for the time value of the start delay.) 5) The next thread is issued. 6) and so forth.

Specify Target Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before starting an I/O to the next target. This requires multiple targets in the test plan.

Retry Failed I/O – Select the check box and specify the number retries for failed I/Os.

Retry Delay – When **Retry Failed I/O** is selected, edit the time value in Hours, Minutes, and Seconds to set the delay between retries.

I/O Behavior dropdown list

Keep File Handles Open Between I/Os – Select this option to keep the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. This option is ignored for 'sock'.

Close File Handles After Every I/O – Select this option to close the target file descriptor (handle) and re-open after each FOP.

Keep File Handles and Flush Every I/O – Select this option to sync (flush) after each FOP. This makes a request to the operating system to commit all written data to the target device, but it may not bypass the device cache. This option is ignored for 'sock'.

Triggering – Set up the triggers during your test. It instructs the tools to send a write I/O to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. It also generates additional log files that are extremely useful in regards to debug and analysis. The data value to trigger on occurs in the first two words of the data frame. The options associated with this switch are:

Disable Triggering – disables the triggering option.

Write 0xCACACACA 0xCACACACA on Data Corruption
Write 0xCACACACA 0xDEADBEEF on I/O Error – Writes 0xCACACACA 0xCACACACA for data corruption trigger and 0xCACACACA 0xDEADBEEF for I/O error trigger.

Write 0xDEADDEAD 0xDEADDEAD on Data Corruption
Write 0xDEADDEAD 0xDEADBEEF on I/O Error – Writes 0xDEADDEAD 0xDEADDEAD for data corruption trigger, and 0xDEADDEAD 0xDEADBEEF for I/O error trigger.

Select any of the two previous options to continue testing if data corruption or I/O error are detected, but generate a trigger. A write command is sent to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. Additional log files are also generated that are extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer (or phase or write). Because the data frame is consistent with FC or serial storage, but not parallel storage testing, the trigger can be used to catch I/O disruptions on an analyzer. The I/O trigger is sent when a halt or stuck I/O is detected.

Stop Testing Immediately - No Trigger Written – Exits the application immediately and no trigger is written.

Write Default (0xCACACACA) Trigger and Exit – Writes default (0xCACACACA) trigger and exits immediately.

Trigger External Application – Executes external application when triggers are detected. Enter the application in the **Application** text box. Enter the arguments to use when running an external Application in the **Arguments** text box.

This last option can be used to trigger the Xgig Analyzer to start (trigger) or stop capture. For example, to trigger the Analyzer operating in the domain “My Domain (1,1,1) XGIG01001234”, set the application to triggeranalyzer.cmd and enter the arguments as “My Domain(1,1,1)” XGIG01001234 in the **Arguments** text box.

Target Offsets –

Override Default/Test Plan Offsets – Select this check box to override the MLTT default device base offset setting. By default, I/O starts at a 1MB offset on the specified device.

Use Default Offset – uses the default offset.

Use a Shared Offset – allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently.

Specify Start Offset – specifies the starting offset number. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target device.

Error Handlers – allow you to specify error handlers.

To add an error handler:

1. Click **Add**.
2. Select the **Handled Error Value** from the drop-down list.
3. Select the **Label Value as** an **Error**, **Warning**, or **Information**.
4. Select the **Trigger** behavior from the drop-down menu.
5. Select **Specify Trigger Pattern** to enter the trigger pattern.
6. Specify the **Exit Mode** from the drop-down list.
7. Select **Specify Retries** to set number of retries for that error handler.

To remove an error handler:

1. Select the error handler from the **Error Handlers** list.
2. Click **Remove**.

Advanced I/O Tab

The **Advanced I/O** tab allows you to specify custom read/writes.

Advanced Read / Write Mix

When you make changes in this area, the **Read / Write Mix** settings on the **I/O Payload** tab are rendered void and as such this area is not displayed on the tab.

To specify a custom read/write mix:

1. Select the **Specify Custom Read / Write Mix** check box.
2. Click **Add** to add a new custom read/write mix.
3. Select the newly added custom read/write mix from the list.

4 Click the **Rebalance to 100%** button to automatically change the access percentage values of the custom read/write mixes to total 100%.

5 Use the options in the **Read/Write Specification** panel to customize the read/write mix.

To remove a custom read/write mix:

1 Select the custom read/write mix from the list.

2 Click **Remove**.

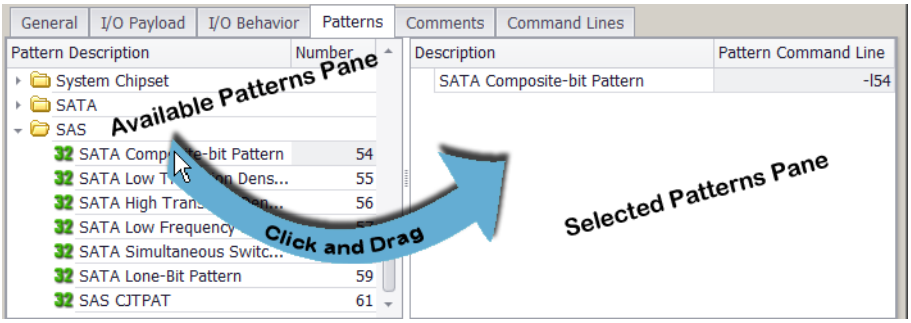
Patterns Tab

The **Patterns** tab allows you to add specific patterns to the test, such as flip/flop patterns, inverted patterns, pattern reversals, data scrambling, or unique data patterns.

Available Patterns – This pane on the upper left of the tab page lists the patterns available for the tests. Several folders are displayed for each available category. Click on the plus/minus sign beside the category type folder to show the list of patterns available in that category.

Selected Patterns – This pane on the upper left of the tab page shows the selected patterns for the current test configuration. This pane displays the test description, the test number, and the command line for the test.

Figure 54: Available Patterns and Selected Patterns



To add a pattern for the current test configuration, click the desired pattern name in the **Available Patterns** pane and drag it into the **Selected Patterns** pane. You can also click a folder and drag it to the **Selected Patterns** pane to add all of the patterns in the folder.

To remove a pattern from the **Selected Patterns** pane, select it and press **Delete** on your keyboard. You can select multiple patterns to delete using the keyboard's Shift or Ctrl buttons.

Pattern Editor Tab

This tab shows the description and the settings for the selected pattern in the **Selected Patterns** pane. The description of the selected pattern is displayed directly beneath the tab name. The options change for the various types of patterns. Select the options for the specific test being performed.

Invert Patterns – This option causes a bit inversion of the data pattern with each transition cycle and is often used to create bit-blink variations over bus architectures.

Use Pattern Reversals – Most data patterns reverse after each FOP (forward, then backward). In some tests (multi-mode, for example), data pattern reversals may look like false data corruptions. Reversals should be allowed anytime data comparisons are being performed as a means of insuring that stale data is not being read.

Reset Pattern Each Cycle – This option causes a “flip/flop” variation to occur within the blinking data pattern. The term “flip/flop” means that the pattern starts at an initial value, inverts (blinks) the value, returns to the initial value, then walks a bit and repeats the sequence.

Scramble Data – Shows options to pre-scramble data patterns according to SAS or SATA specifications. When these patterns are written by MLTT, hardware scrambling will have the effect of de-scrambling the data into the desired pattern. This is an effective means of signal integrity testing on these architectures when combined with the Fibre Channel data patterns. The SAS and SATA options will automatically use default frame data lengths for the scrambler reset. The data length/reset interval can be overridden by specifying the data length in bytes.

No Data Scrambling – No scrambling of data patterns.

SAS Data Scrambling – Pre-scrambles data patterns according to SAS specifications.

SATA Data Scrambling – Pre-scrambles data patterns according to SATA specifications.

Scramble Reset Interval – When you specify a reset interval, you override the data length/reset interval for the scrambled pattern. Select the **Specify Reset Interval** check box and then edit the number to specify the data length. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Data Pattern Cycle Length – The cycle length indicates the number of times to repeat each cycle of a data pattern before moving to the next unit. Select the **Cycle Data Pattern** check box and set the cycle length. In general, the unit of data pattern refers to its length in bytes or bits. An example use of this option is to run an 8-bit pattern four times to produce, effectively, a 32-bit pattern. In this case, each byte is run four times before moving on to the next byte.

Phase shift – This pertains to most blinking data patterns. If the **Use Default Phase Shift** or **Specify Phase Cycle Length** options are selected, the data pattern shifts at the specified cycle length, such that the square wave, created by the on/off bits in the blinking byte values, reverses. The frequency of this shift is determined by the cycle length setting. Cycle length multiplied by pattern length determines the shift frequency.

To use this feature, select either **Use Default Phase Shift** or **Phase Cycle Length**. When **Phase Cycle Length** is selected, enter the number of cycles to run before doing the phase shift. Change this value by entering a value in the text box or using the numeric spinner.

Data Pattern Specification – Allows you to specify a static value to use as the static repeating pattern if you don’t want to use with the default. The default pattern for those is to use the thread number for the pattern value. This is repeated continuously. Change this value using the numeric spinner.

Random Seed – Specifies an initialization value for the pseudorandom number generator used to generate a random pattern.

Walking Bit Options – This walks an opposing bit across the sequence when the pattern is a blinking pattern.

Note: Each of the Walking Bit options are shown in Table 11. The opposing bits are shaded in the table so the walking effect can be seen easily.

Table 11: Walking Bits Using an 8-bit Blinking Example

Do Not Use Walking Bits	Walk Bits on 'ON' Cycle	Walk Bits on 'OFF' Cycle	Walk Bits on Both Cycle
00000000	00000000	10000000	10000000
11111111	01111111	11111111	01111111
00000000	00000000	01000000	01000000
11111111	10111111	11111111	10111111
00000000	00000000	00100000	00100000
11111111	11011111	11111111	11011111
00000000	00000000	00010000	00010000
11111111	11101111	11111111	11101111
00000000	00000000	00001000	00001000
11111111	11110111	11111111	11110111
00000000	00000000	00000100	00000100
11111111	11111011	11111111	11111011
00000000	00000000	00000010	00000010
11111111	11111101	11111111	11111101
00000000	00000000	00000001	00000001
11111111	11111110	11111111	11111110

Do Not Use Walking Bits – Walking bits are not used.

Walk Bits on 'ON' Cycle – Walking bits only walks "0" across the "1's" cycle.

Walk Bits on 'OFF' Cycle – Walking bits only walks "1" across the "0's" cycle.

Hold Pattern for cycles before walking – Keeps the walking bit in its position for the specified number of cycles before advancing the bit to its the next position. The number of cycles is maintained at each of the bit's walking positions. Change this value by entering a value in the text box or using the numeric spinner.

Set Blink Length – Sets the length of the "ON" ('1') bits. Change this value by entering a value in the text box or using the numeric spinner.

Hexadecimal Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in hexadecimal format.

Binary Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in binary format.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option. This is useful when using any of the modes that create multiple tests with one configuration file (i.e. ranged values, cycle read/write modes, or multiple data patterns, etc.)

TCP App Simulation Configuration Editor

The TCP App Simulation configuration editor allows you edit settings for emulating TCP traffic by using transactional data instead of read/write mixes.

To open the editor, double-click the TCP App Simulation configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: Some of the options on the editor may be grayed out based on the methodology selected or other option dependencies. The editor opens in the same mode, either in basic or advanced, depending on the mode when it was closed.

The editor has six tabs for specifying testing parameters:

- General (See page 109)
- I/O Payload (See page 110)
- I/O Behavior (See page 112)
- Patterns (See page 114)
- Comments (See page 116)
- Command Lines (See page 117)

The description of each of these tabs and its parameters in the following pages.

General Tab

The **General** tab shows the following settings for this configuration editor.

Startup Options –

Initial Sample Delay – Sets the delay time to wait before starting the testing. This delay allows devices the allotted time to get setup. The **H** (Hour), **M** (Minute), and **S** (Second) numeric spinners allow you to delay the start of the testing up to 23 hours, 59 minutes, and 59 seconds.

Throughput Limitation – sets the limitations for the maximum I/O throughput

Infinity – sets the maximum I/O throughput to have no limitations.

Every Target – sets the **Max I/O Throughput** value to apply to every target.

Every Thread – sets the **Max I/O Throughput** value to apply to every thread.

Max I/O Throughput – provides the maximum I/O throughput limitation value that is applied to every target or thread.

External Application – runs any application of your choosing after every test has been run. For example, if you want to run an independent application to collect data (such as a log file) from a device, you can use this to start the application.

The **Run External Application After Test** check box must be selected before you can select the application and wait times.

Application allows for you to browse to and select the desired application.

Wait for External Application allows you to input a period to wait for the application to start to run. Tests tools will continue testing once the external application begins running.

I/O Payload Tab

The **I/O Payload** tab shows the basic parameters for a Test Tool test, such as the testing threads, requests from the client, responses from the server side, and the logging level selections. Queue depth is also available after other parameters on the tab are changed.

Performance performance mode – Select the **Enable Performance mode** check box to enable the performance mode. This mode increases the speed of testing by optimizing the use of internal memory buffers. When this check box is selected, the following settings are automatically set:

Table 12: Performance Mode Settings

GUI Setting	Command Line Setting
I/O Marking and Signing is set to **No I/O Markings** No I/O signatures are applied to each sector of every write.	`-u` (page 177)
Data Compare Mode is set to **Disable Data Comparisons** Data comparisons are turned off.	`-n` (page 177)
I/O Behavior is set to **Keep File Handles Open Between I/Os** Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP.	`-o` (page 166)
Use Pattern Reversals check box is not selected (cleared) Leaves the data patterns reversal after each FOP (forward, then backward) turned off.	`-N` (page 174)

Testing Threads – Set the threads for testing. Each thread executes a single I/O at a time, with each thread starting at a different base offset. The number of threads successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

You can also set a range of threads to test by selecting the **Test a Range of Threads** check box. Set the **Thread Count Start** and the **Thread Count End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the tested threads through the specified range from the start to the end. Refer to “Test a Range Controls” on page 65 for more information about the **Adding** and **Multiplying** selections.

Queue Depth – (available after other tab parameters are changed) Set the queue depth.

The queue depth is the maximum number of current I/Os to execute in a single worker thread. The value of queue depth successfully run is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the queue depth by entering the value or clicking the **Queue Depth** numeric spinner.

You can also set a range for the queue depth by selecting the **Test a Range of Queue Depths** check box. Set the **Queue Depth Start** and the **Queue Depth End** values and then set the **Adding** or **Multiplying** values. The adding/multiplying value increments the queue depth through the specified range from the start to the end. Refer to "Test a Range Controls" on page 65 for more information about the **Adding** and **Multiplying** selections.

The **Queue Depth** has two additional options, **Keep Queue Depth Static** and **Strict Sequential**.

Select **Keep Queue Depth Static** to add `-m16` option to the command line to use continuous queuing. If this option is not selected, by default burst queuing will be used.

Select **Strict Sequential** to add the -m1 option to the command line which will make the test have continuous queuing and strict sequential access.

Requests (from client side) - Add or remove requests from the client side.

To add a request:

1. Click **Add**.
2. Adjust the settings for the request by specifying size and traffic percentage.
3. Select the **Rebalance to 100%** button to balance the traffic across the requests.

To remove a request:

1. Select the request from the list.
2. Click **Remove**.

Responses (from server side) - Add or remove responses from the server side.

To add a response:

1. Click **Add**.
2. Adjust the settings for the response by specifying size and traffic percentage.
3. Select the **Rebalance to 100%** button to balance the traffic across the responses.

To remove a response:

1. Select the response from the list.
2. Click **Remove**.

Logging Level – Select the type of logging you want for the configuration to indicate the level of information to be posted to the log file. The default option posts the maximum amount of information which is helpful for analyzing errors.

Standard logfile generation/output – Default option, includes detailed headers and console performance output.

Outputs to logfile in test performance format (minimal logging) – Removes headers and logs performance output only.

No outputs to logfile, minimal screen outputs, PRF log summary – No .log file generated, and logs minimal screen output.

Disable CSV log – Standard logging, but .csv file will not be created.

Single line output with system name, performance, and errors – Includes system name and other details on single output lines for easier importation or parsing.

Disable completion statistics in PRF file – Disables completion calculations and output. In IOPS intensive tests where the CPU is heavily taxed, using this option may result in a slight performance gain.

Enable logging of informational events in Windows event log – This option will send informational output to Windows event log, such as test start and stop details.

Command Line – As options are selected, the equivalent command line settings appear in the textbox. See "Using the Command Line Switches" on page 123 for details of the command line settings.

I/O Behavior Tab

The **I/O Behavior** tab allows you to specify Transactions, I/O Behavior, Triggering, and Error Handling options based on test results.

Transactions – Select the options for transactions in this panel.

1. Select **Specify the number of transactions per connection** to set minimum and maximum number of transactions.
2. Set the **Minimum transactions**.
3. Set the **Maximum transactions**. As an alternative, you can select the **Use minimum** check box to use the minimum transactions number as the maximum value also.

I/O Behavior – Modify the behavioral aspects of I/O in a test.

Specify Burst Interval – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the burst interval duration.

Specify Thread Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before issuing the next thread. This requires a multi-threaded test definition.

For the start delay: 1) The first thread is issued. 2) There is a pause (for the time value of the start delay.) 3) The next thread is issued. 4) There is a pause (for the time value of the start delay.) 5) The next thread is issued. 6) and so forth.

Specify Target Start Delay – Select the check box and edit the time value in Hours, Minutes, and Seconds to set the delay before starting an I/O to the next target. This requires multiple targets in the test plan.

Retry Failed I/O – Select the check box and specify the number retries for failed I/Os.

Retry Delay – When **Retry Failed I/O** is selected, edit the time value in Hours, Minutes, and Seconds to set the delay between retries.

Triggering – Set up the triggers during your test. It instructs the tools to send a write I/O to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. It also generates additional log files that are extremely useful in regards to debug and analysis. The data value to trigger on occurs in the first two words of the data frame. The options associated with this switch are:

Disable Triggering – disables the triggering option.

Write 0xCACACACA 0xCACACACA on Data Corruption
Write 0xCACACACA 0xDEADBEEF on I/O Error – Writes 0xCACACACA 0xCACACACA for data corruption trigger and 0xCACACACA 0xDEADBEEF for I/O error trigger.

Writes 0xDEADDEAD 0xDEADDEAD on Data Corruption
Write 0xDEADDEAD 0xDEADBEEF on I/O Error – Writes 0xDEADDEAD 0xDEADDEAD for data corruption trigger, and 0xDEADDEAD 0xDEADBEEF for I/O error trigger.

Select any of the two previous options to continue testing if data corruption or I/O error are detected, but generate a trigger. A write command is sent to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. Additional log files are also generated that are extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer (or phase or write). Because the data frame is consistent with FC or serial storage, but not parallel storage testing, the trigger can be used to catch I/O disruptions on an analyzer. The I/O trigger is sent when a halt or stuck I/O is detected.

Stop Testing Immediately - No Trigger Written – Exits the application immediately and no trigger is written.

Write Default (0xCACACACA) Trigger and Exit – Writes default (0xCACACACA) trigger and exits immediately.

Trigger External Application – Executes external application when triggers are detected. Enter the application in the **Application** text box. Enter the arguments to use when running an external Application in the **Arguments** text box.

This last option can be used to trigger the Xgig Analyzer to start (trigger) or stop capture. For example, to trigger the Analyzer operating in the domain “My Domain (1,1,1) XGIG01001234”, set the application to triggeranalyzer.cmd and enter the arguments as “My Domain(1,1,1)” XGIG01001234 in the **Arguments** text box.

Error Handlers – allow you to specify error handlers.

To add an error handler:

1. Click **Add**.
2. Select the **Handled Error Value** from the drop-down list.
3. Select the **Label Value as** an **Error**, **Warning**, or **Information**.
4. Select the **Trigger** behavior from the drop-down menu.
5. Select **Specify Trigger Pattern** to enter the trigger pattern.
6. Specify the **Exit Mode** from the drop-down list.
7. Select **Specify Retries** to set number of retries for that error handler.

To remove an error handler:

1. Select the error handler from the **Error Handlers** list.
2. Click **Remove**.

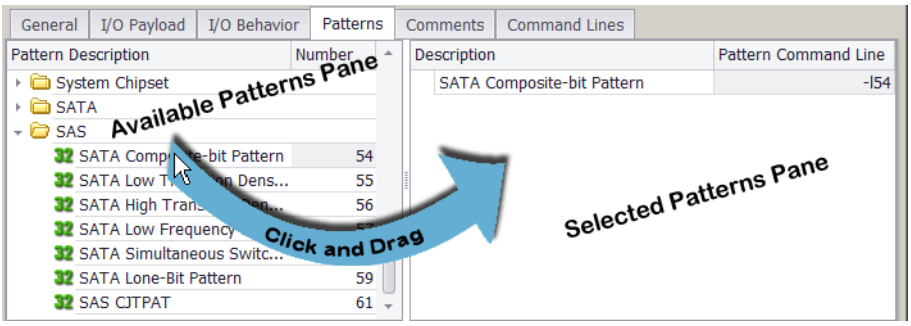
Patterns Tab

The **Patterns** tab allows you to add specific patterns to the test, such as flip/flop patterns, inverted patterns, pattern reversals, data scrambling, or unique data patterns.

Available Patterns – This pane on the upper left of the tab page lists the patterns available for the tests. Several folders are displayed for each available category. Click on the plus/minus sign beside the category type folder to show the list of patterns available in that category.

Selected Patterns – This pane on the upper left of the tab page shows the selected patterns for the current test configuration. This pane displays the test description, the test number, and the command line for the test.

Figure 55: Available Patterns and Selected Patterns



To add a pattern for the current test configuration, click the desired pattern name in the **Available Patterns** pane and drag it into the **Selected Patterns** pane. You can also click a folder and drag it to the **Selected Patterns** pane to add all of the patterns in the folder.

To remove a pattern from the **Selected Patterns** pane, select it and press **Delete** on your keyboard. You can select multiple patterns to delete using the keyboard's Shift or Ctrl buttons.

Pattern Editor Tab

This tab shows the description and the settings for the selected pattern in the **Selected Patterns** pane. The description of the selected pattern is displayed directly beneath the tab name. The options change for the various types of patterns. Select the options for the specific test being performed.

Invert Patterns – This option causes a bit inversion of the data pattern with each transition cycle and is often used to create bit-blink variations over bus architectures.

Use Pattern Reversals – Most data patterns reverse after each FOP (forward, then backward). In some tests (multi-mode, for example), data pattern reversals may look like false data corruptions. Reversals should be allowed anytime data comparisons are being performed as a means of insuring that stale data is not being read.

Reset Pattern Each Cycle – This option causes a "flip/flop" variation to occur within the blinking data pattern. The term "flip/flop" means that the pattern starts at an initial value, inverts (blinks) the value, returns to the initial value, then walks a bit and repeats the sequence.

Scramble Data – Shows options to pre-scramble data patterns according to SAS or SATA specifications. When these patterns are written by MLTT, hardware scrambling will have the effect of de-scrambling the data into the desired pattern. This is an effective means of signal integrity testing on these architectures when combined with the Fibre Channel data patterns. The SAS and SATA options will automatically use default frame data lengths for the scrambler reset. The data length/reset interval can be overridden by specifying the data length in bytes.

No Data Scrambling – No scrambling of data patterns.

SAS Data Scrambling – Pre-scrambles data patterns according to SAS specifications.

SATA Data Scrambling – Pre-scrambles data patterns according to SATA specifications.

Scramble Reset Interval – When you specify a reset interval, you override the data length/reset interval for the scrambled pattern. Select the **Specify Reset Interval** check box and then edit the number to specify the data length. The size can be set to **Bytes**, **KB**, **MB**, **GB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target device; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Data Pattern Cycle Length – The cycle length indicates the number of times to repeat each cycle of a data pattern before moving to the next unit. Select the **Cycle Data Pattern** check box and set the cycle length. In general, the unit of data pattern refers to its length in bytes or bits. An example use of this option is to run an 8-bit pattern four times to produce, effectively, a 32-bit pattern. In this case, each byte is run four times before moving on to the next byte.

Phase shift – This pertains to most blinking data patterns. If the **Use Default Phase Shift** or **Specify Phase Cycle Length** options are selected, the data pattern shifts at the specified cycle length, such that the square wave, created by the on/off bits in the blinking byte values, reverses. The frequency of this shift is determined by the cycle length setting. Cycle length multiplied by pattern length determines the shift frequency.

To use this feature, select either **Use Default Phase Shift** or **Phase Cycle Length**. When **Phase Cycle Length** is selected, enter the number of cycles to run before doing the phase shift. Change this value by entering a value in the text box or using the numeric spinner.

Data Pattern Specification – Allows you to specify a static value to use as the static repeating pattern if you don't want to use with the default. The default pattern for those is to use the thread number for the pattern value. This is repeated continuously. Change this value using the numeric spinner.

Random Seed – Specifies an initialization value for the pseudorandom number generator used to generate a random pattern.

Walking Bit Options – This walks an opposing bit across the sequence when the pattern is a blinking pattern.

Note: Each of the Walking Bit options are shown in Table 13. The opposing bits are shaded in the table so the walking effect can be seen easily.

Table 13: Walking Bits Using an 8-bit Blinking Example

Do Not Use Walking Bits	Walk Bits on 'ON' Cycle	Walk Bits on 'OFF' Cycle	Walk Bits on Both Cycle
00000000	00000000	10000000	10000000
11111111	01111111	11111111	01111111
00000000	00000000	01000000	01000000
11111111	10111111	11111111	10111111
00000000	00000000	00100000	00100000
11111111	11011111	11111111	11011111
00000000	00000000	00010000	00010000
11111111	11101111	11111111	11101111
00000000	00000000	00001000	00001000
11111111	11110111	11111111	11110111
00000000	00000000	00000100	00000100
11111111	11111011	11111111	11111011
00000000	00000000	00000010	00000010
11111111	11111101	11111111	11111101
00000000	00000000	00000001	00000001
11111111	11111110	11111111	11111110

Do Not Use Walking Bits – Walking bits are not used.

Walk Bits on 'ON' Cycle – Walking bits only walks "0" across the "1's" cycle.

Walk Bits on 'OFF' Cycle – Walking bits only walks "1" across the "0's" cycle.

Walk Bits on Both Cycle – Walking bits are walked across both cycles.

Hold Pattern for cycles before walking – Keeps the walking bit in its position for the specified number of cycles before advancing the bit to its the next position. The number of cycles is maintained at each of the bit's walking positions. Change this value by entering a value in the text box or using the numeric spinner.

Set Blink Length – Sets the length of the "ON" ('1') bits. Change this value by entering a value in the text box or using the numeric spinner.

Hexadecimal Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in hexadecimal format.

Binary Preview Tab

This tab displays the selected data pattern in the **Selected Patterns** pane in binary format.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option. This is useful when using any of the modes that create multiple tests with one configuration file (i.e. ranged values, cycle read/write modes, or multiple data patterns, etc.)

Network CLI Configuration Editor

The Network CLI configuration editor saves sock command lines so the command lines can be sent from the configuration editor.

To open the editor, double-click the Network CLI configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: The sock commands sent from this editor do not have error checking. If invalid commands are entered and sent, these commands are ignored and are not reported.

The editor has two tabs for specifying testing parameters and information:

- Command Line
- Comments

Command Line Tab

The **Command Line** tab allows you to enter sock commands. This is useful when you want to run a command from the GUI.

The **Paste** button pastes a previously-copied command line to the configuration editor.

The **Copy** button copies the configuration editor's command line once it is selected.

For pain and maim commands, refer to "Storage CLI Configuration Editor" on page 97.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

SSD Secure Erase Configuration Editor

The SSD Secure Erase configuration editor erases the data on a Solid State Drive (SSD) leaving it in a *clean* state after the test process using the configuration is complete.

To open the editor, double-click the Solid State Drive (SSD) Secure Erase configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: The Individual Test Setup settings in the Planning Group Editor (shown in Figure 26 on page 40) and the Test Plan Editor (shown in Figure 27 on page 42) have no effect on the SSD Secure Erase configuration.

The editor has three tabs for specifying testing parameters and information:

- SE Operation
- Command Lines
- Comments

SE Operation Tab

The **SE Operation** tab allows you to select the type of secure erase that you want to perform. Optionally, you may provide a time-out value to end the test if the erasure operation has not completed in the specified time.

Mode – sets the erasure level for the solid state drive used as the target. The erasure levels are:

Standard erase for ATA or format unit for SCSI –

- For ATA devices, this selection goes through and marks each cell as empty.
- For SCSI devices, this selection requests that the device server format the medium into application accessible logical blocks.

Enhanced erase for ATA or sanitize for SCSI –

- For ATA devices, this selection writes predetermined data patterns (set by the manufacturer) to all user data areas, including sectors that are no longer in use due to reallocation.
- For SCSI devices, this selection performs a format unit command then performs a pattern overwrite of the accessible logical blocks.

Time Out – Sets the **Max allowed time** (a time-out value) which will end the test if the erasure operation has not completed in the specified time. If the erase operation is not completed, the program will exit with the TIMEOUT_ERROR program exit code. If no time out value is set (**Max allowed time** is left in the default value of 0H 0M 0S), there is no active time out setting and the erasure operation is allowed to run until it is complete.

You can set the maximum time allowed by entering the value or clicking the **Max allowed time** numeric spinner.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option.

SSD Trim Configuration Editor

The SSD Trim configuration erases specified data blocks. It may be run as a target Solid State Drive (SSD) pre-conditioning step before running I/O tests.

To open the editor, double-click the Solid State Drive (SSD) Trim configuration that you added to the **User Configuration** folder in the **Configurations** area. You may also use one of the methods discussed in "Using the Configuration Editors within the GUI" on page 62.

Note: The Individual Test Setup settings in the Planning Group Editor (shown in Figure 26 on page 40) and the Test Plan Editor (shown in Figure 27 on page 42) have no effect on the SSD Trim configuration.

The editor has three tabs for specifying testing parameters and information:

- Trim Operation
- Command Lines
- Comments

Trim Tab

The **Trim** tab allows you to set the parameters for erasing specified blocks from the target SSD.

Threads – sets the number of threads on the SSD being pre-conditioned for testing. A trim command is executed for each specified thread. The number of threads is dependent on the available memory resource. The practical limitation also depends on the target capabilities.

You can set the thread count (number of threads) by entering the value or clicking the **Thread Count** numeric spinner.

Testing Sizes – sets the data block sizes to be erased. **Testing Sizes** settings include:

Specify Testing Area – Select this button and specify the file size or disk area to erase per thread.

The size can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, or **Units** from the drop-down menu. In this case, **Units** refers to a block size that is reported back from the target disk; for example, a block sizes of 512 bytes or some other block size that is standard to the target.

Test Using the Entire Target – Select this check box to erase the entire data area on the target SSD.

Target Offsets – begins the trim/erase operation at the specified offset.

Starting Offset – specifies the starting offset number of the erase operation. Select from the dropdown menu the unit of the value you specified. The offset value must be a multiple of the logical block size of the target disk.

Time Out – Sets the **Max allowed time** (a time-out value) which will end the test if the trim/erase operation has not completed in the specified time. If the operation is not completed, the program will exit with the TIMEOUT_ERROR program exit code. If no time out value is set (**Max allowed time** is left in the default value of 0H 0M 0S), there is no active time out setting and the erasure operation is allowed to run until it is complete.

You can set the maximum time allowed by entering the value or clicking the **Max allowed time** numeric spinner.

Comments Tab

Enter your comments for the configuration in the comment box of the **Comments** tab.

Command Lines Tab

The **Command Lines** tab allows you to display a listing of the command line commands required by the GUI configuration. Select the **List Command Lines** button to display the listing. The command line listing can be copied by selecting the command lines, right-clicking on the selection, and choosing the **Copy** option.

Chapter 4

Using the Command Line Switches

In this chapter:

Syntax

You use switches at the command line to enter the parameters you want for your test. A basic command line entry contains the following:

```
Application name, target, I/O size, file size, queue depth or thread count, data pattern
```

The following example shows the syntax for a test:

```
pain -f\\.\physicaldrive2 -b512k 100 -t8 -125
```

where:

`pain` is the application name.

`-f` is the target (see “Target Specification” on page 127)

`\\.\physicaldrive2` is the target device.

`-b512k` is the buffer size (see “I/O Size” on page 129)

`100` is the file size (see “File Size” on page 130).

`-t8` is the thread count (see “Thread Count” on page 132)

`-125` is the data pattern (see “Data Pattern” on page 134)

Important: You can specify the command line switches in any order. All Medusa Labs Test Tools (MLTT) switches are case sensitive.

Basic Switches

This section describes the switches you use for most test runs. The switches are listed by function.

Target Specification

```
-f   Target
```

specifies the desired target

I/O Size

```
-b   Buffer size
```

specifies the buffer size for each I/O

File Size

```
file_size
```

specifies the desired “file” size as a number

Queue Depth

```
-Q   Queue_Depth (Maim only)
```

specifies the maximum number of outstanding I/Os when using Maim

Thread Count

```
-t   Thread Count
```

specifies the number of worker threads when using Pain

Data Pattern

```
-l Data Pattern
```

specifies the desired data pattern number

Online Help

```
-h   Online Help
```

displays the online help

Default Session ID

`-A  Default Session ID`

overrides the default session ID in data signatures with the specified 16-bit ID

Timestamps

`-U  I/O Signature Timestamp Units`

sets timestamps in I/O signature in seconds or in milliseconds

License Client Operation

`-Z  License Client Operation`

manages remote license operation

Seconds Between Performance Samples

`-Y  Seconds Between Performance Samples`

specifies the number of seconds between performance samples displayed on the screen and printed to log files.

Target Specification

-f Target

Usage:

`-ftarget`

Description:

Use `-f` to specify the desired target. The target can be a file, logical drive, or physical drive that resides in the host system or is externally attached via SCSI, USB, FireWire, LAN, SAN, and others.

Also, when using sock, the target may specify the hostname or an IP (or IPv6) address of a peer for TCP/IP network I/O.

Note: If the switch is not specified, one file of the specified size is created in the current directory by each worker thread.

Default:

If no target is specified, each worker thread creates a file in the current directory.

Examples:

Physical: `-f\\.\physicaldrive1`

Logical: `-f\\.\g:`

File: `-fg:\file1.dat`

Linux device: `-f/dev/sdc`

Solaris device: `-f/dev/rdsk/c1t1d0s2`

TCP/IP peer (sock): `-fhostname` or `-f10.23.1.101` or `-f10.23.1.80:10.23.1.101` (where `10.23.1.80` is a specific local IP if there is more than one network interface to choose from). When specifying an IPv6 address pair, use '-' to separate the local and remote addresses (e.g. `-ffe80::c62c:3ff:fe08:a66c%en0-fe80::221:9bff:fe50:90ec`). For a link-local IPv6 address pair, the scope ID of the outgoing interface must be appended to the local address (e.g. `"%en0"` or `"%5"`).

The tools also support a multi-target mode, where multiple targets can be accessed in a single process. The Catapult -t switch option performs this automatically. See "-t Multi-target mode" on page 226. Multiple targets may also be specified manually in one of several manners:

- Create a text file called "targets.dat" that contains desired targets, one per line. Catapult can create this file for you. For example:

  ```
  catapult -p -t
  ```

  Then pass this file name, with path if necessary, to pain or maim with the -f switch.

  ```
  pain -ftargets.dat
  ```

- You can also specify multiple targets on the command line, separated by commas. For example:

  ```
  pain -f\\.\physicaldrive1,\\.\physicaldrive2
  ```

- You can also use a prefix system, where a common prefix is terminated with a semi-colon, followed by suffixes that are comma separated. For example:

  ```
  pain -f\\.\physicaldrive;1,2,3
  ```

- Generate TCP/IP I/O to `10.23.1.101` and `10.23.1.102` from `10.23.1.80`.

  ```
  sock -f"10.23.1.80:10.23.1.;101,102"
  ```

Note: on Unix systems, the shell interprets ';' as the command separation character; therefore, the target name should be quoted. For example, the shell interprets the following:

pain -f/dev/sd;b,c,d

as a sequence of the two commands shown below:

pain -f/dev/sd

b,c,d

To prevent such errors, the target specification must have quotes added as shown:

pain -f"/dev/sd;b,c,d"

Warning: Physical and logical drive access is destructive!
Existing data WILL be overwritten.

I/O Size

-b Buffer size

Usage:

`-bbuffer_size[units]`

Description:

Use -b to specify the buffer size for each I/O. This equates to the I/O block size, from the application level. You can specify the buffer size in bytes or use a numeric value and unit designator:

`m` = megabytes

`k` = kilobytes

`b` = bytes (default)

u = LB (logical block) units, usually 512 bytes

Note: For physical drive targets, buffer size must be a multiple of device's logical block size (usually 512 bytes.) For other types of targets (such as a regular file), the buffer size must be a multiple of 512 bytes.

Examples:

`-b1m` = 1 megabyte (1048576 bytes.)

`-b4k` = 4 kilobytes (4096 bytes.)

`-b8192` = 8 kilobytes

Default:

The default buffer size in MLTT is 64 kilobytes (65536 bytes).

File Size

file_size

Usage:

file_size

Description:

The desired “file” size is specified as a number, with no preceding switch argument. You can specify the size in bytes or use a numeric value and unit designator:

`g` = gigabytes

`m` = megabytes (default)

`k` = kilobytes

u = LB (logical block) size units, usually 512 bytes

Examples:

```
1g
100m
512k
```

The file size must be at least the same as the I/O size, or a multiple of the I/O size.

Maximum file size is as allowed by the operating system.

The file size can apply to an actual file in file system based testing or the extent of linear space to utilize on a logical or physical drive. Note that file size will be utilized per thread (that is, each worker thread in our thread-based tools will utilize the extent specified by the file size – 8 threads x 100m [`-t8 100`] would equal 800 megabytes total.) The default file size varies in each tool.

Default:

The default file size per thread in Pain is 4MB.

The default file size for the single worker thread in Maim is 10MB.

Queue Depth

-Q Queue_Depth (Maim only)

Usage:

`-Qqueue_depth`

Description:

Use `-Q` to specify the maximum number of outstanding I/Os (queue depth) when using Maim. The maximum number of outstanding (pending) I/Os is dependent on the operating system and memory resources. The practical limitation also depends on the target capabilities.

Example:

The following switch specifies a queue depth of 8 I/Os to be created by the worker thread.

`-Q8`

Default:

The default queue depth is one.

Thread Count

-t Thread Count

Usage:

`-tthread_count [p]`

Description:

Use -t to specify number of worker threads in pain, maim, and sock. In sock, this corresponds to the number of concurrent socket connections per sock process. In pain (synchronous I/O), since each thread dispatches a single I/O at a time, this number roughly correlates to queue depth. In maim (asynchronous I/O), each thread tries to maintain concurrent I/O operations specified by -Q so the potential maximum I/O operations in-flight per target device is the product of the thread count and the queue depth.

Figure 56: Thread Count Command (Threads per Target) Example

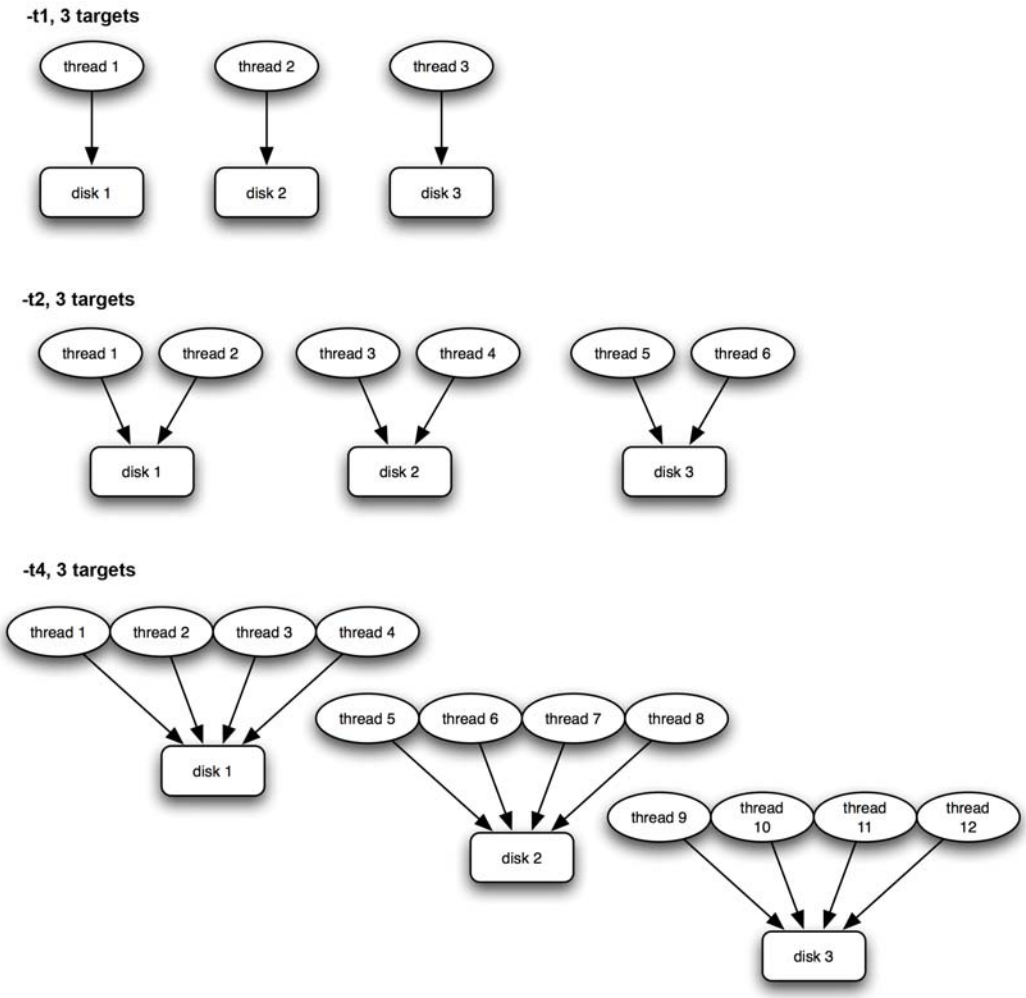
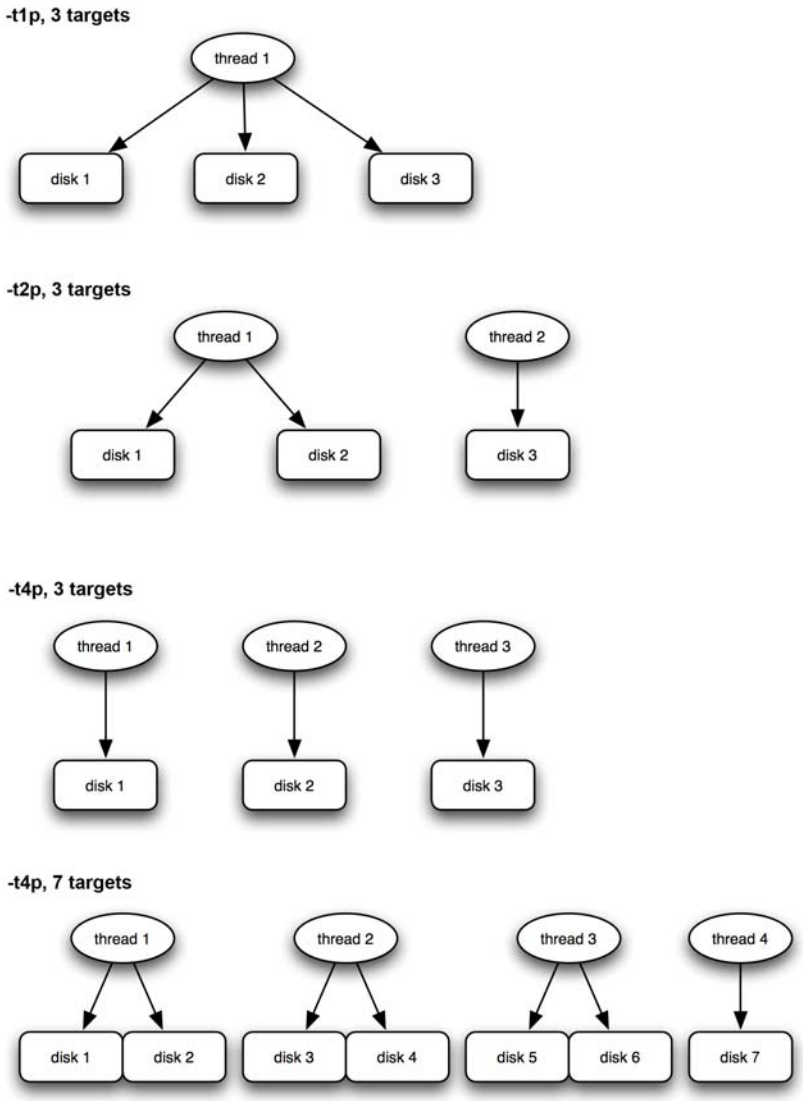


Figure 56 shows examples of normal thread count usage (without 'p' suffix) using 3 targets. For each target, <thread_count> number of I/O threads is created. The total number of I/O threads created in this case is "#targets x <thread_count>".

Figure 57 shows the [p] option with the <thread_count> command. With 'p' used, <thread_count> specifies the maximum number of total I/O threads to use per process. So rather than creating "threads-per-target", it assigns "targets-per-thread". This option is valid only for Maim.

Figure 57: Thread Count Command (Threads Per Process) Example



Note that with the "-t4p, 3 targets" case, even though the <thread_count> is 4, it ends up creating 3 total threads because there are only 3 targets. The actual number of I/O threads created is adjusted to be at most the number of targets in use.

Note: If you specify the special case "-t0p", then the program sets the <thread_count> to the number of available CPUs. For example, if the system has 4 CPUs, then "-t0p" is same as "-t4p". If in that same system you also specify "-T2" (i.e. use only the first 2 CPUs), then "-t0p" is same as "-t2p".

With the 'p' modifier, the total number of I/O threads is at most the specified <thread_count> value, no matter the target count.

Default:

The default thread count is `-t1`.

Data Pattern

-l Data Pattern

Usage:

`-l`*`pattern_number`*

Description:

Use `-l` to specify the desired data pattern number. Data pattern refers to the content of the application payload. Typically, you would want to indicate a specific data pattern for any test involving data or signal integrity. The available data patterns are listed in the command line help and in Appendix A, “Data Pattern Numbers.” Refer to Chapter 6, “Data Pattern Reference” for more details on the use of this switch and other related switches.

Default:

The default data pattern varies in each tool.

Switches by Category

In this section, the command-line switches are described by category. Within each category, the switches are listed in alphabetical order. All command line switches can be divided into the following categories:

Note: Not all switches are available in each tool. Use the online help for a complete listing of switches applicable to each individual tool.

General Switches

The switches described in this section are the general switch commands.

-d Test Duration in Seconds

Usage:

`-dseconds`

Description:

Use `-d` to limit the duration of a test to the specified number of seconds. If `-i` is also specified, then the program terminates upon reaching the first exit condition.

Default:

The default behavior of MLTT is to run until manual intervention or a critical error is encountered.

-i Number of Iterations

Usage:

`-iiterations`

Description:

Use `-i` to limit the run time of a test to the specified number of iterations before exiting. An iteration, called a file operation (FOP), is a complete write and read of an entire file or specified extent on a logical or physical drive. If `-d` is also specified, then the program terminates upon reaching the first exit condition. For non-sequential I/O operations, `-i` has no effect and is ignored.

Default:

The default behavior of MLTT is to run until manual intervention or a critical error is encountered.

-q Control Displayed Information

Usage:

`-qmode_number`

Description:

Use `-q` (Quiet Mode) to control the amount of information printed to the screen and log files. The available mode numbers are:

`0` = Standard log file generation/output (default)

`1` = Outputs to log file in test performance format

`2` = No outputs to log file, minimal screen outputs

`3` = Disable CSV log

`4` = Single line output with system name, performance, and errors

`5` = Disable completion statistics in PRF file

`6` = Enable logging of informational events in Windows event log

In general, it is best to leave this setting at the default in order to have the greatest possible amount of information available in the event an error is encountered.

Default:

0 is the default, which is the standard log file generation and output.

-S Seconds to Delay Between Thread Creation

Usage:

`-S<thread_delay>` or `-S.<target_delay>` or `-S<thread_delay>.<target_delay>`

Description:

Use `-S` to specify the number of `<thread_delay>` seconds to delay between each worker thread's starting I/O, or `<target_delay>` seconds to delay thread starting I/O to each target device, or both use `<thread_delay>` and `<target_delay>`. You might need to use this switch in some scenarios if the system does not tolerate the burst of initial threads.

Default:

The default behavior of MLTT is to launch all worker threads at once.

-T Set I/O Thread/CPU Affinity

Usage:

`-T<number_of_CPUs>`

Description:

Use -T to specify the number of CPUs to use for I/O threads. In addition to limiting the number of CPUs used, this option causes a thread to always run on the same CPU. The number of CPUs specified must be equal to or less than the number of CPUs on the system and equal to or less than the total number of I/O threads.

Examples:

Given a system with 2 quad-core CPUs (8 "logical" CPUs total):

```
pain -t16 -T2
```

The example above specifies that only the first 2 logical CPUs to be used for all 16 I/O threads. In addition, specifying `-T` prevents thread migration among CPUs. Therefore, in this example, the first thread always runs on the first CPU, the second thread always runs on the second CPU, the third thread always runs on the first CPU, and so on.

Given the same 8 CPU system:

```
pain -t8 -T8
```

In this example, all available CPUs are utilized. If `-T` was not specified, the operating system might schedule each I/O thread on different CPUs at different times. Because, `-T` is specified, the first thread is always scheduled on the first CPU, the second thread is always scheduled on the second CPU, and so on.

(Note that ‘-T0’ does not mean thread/CPU affinity is not set. It is reserved for a special meaning that has not been implemented).

The user can also specify -T: with selective CPU numbers. For example: `-T:1,4` or `-T:1-4`.

Default:

If ‘-T’ is not specified in the command line, then ‘-T’ is not set (“do not set thread/CPU affinity”).

-Y Seconds Between Performance Samples

Usage:

```
-Y<seconds>
```

Description:

Use -Y to specify the number of seconds between performance samples displayed on the screen and printed to log files.

Example:

-Y1 logs performance samples once per second to the screen.

Default:

Performance samples are taken at 5 second intervals by default.

--sample-delay Specify Sample Delay

Usage:

```
--sample-delay=<seconds>
```

Description:

Use --sample-delay to specify the delay performance sample collection by specified number of 'seconds'.

Default:

'--sample-delay=0'

-h Online Help

Usage:

```
-h
```

Description:

Use the -h switch to display online help.

Note: -? can be used as an alternative to using the -h command to display the online help.

Example:

-h

This example would show the online help.

-h -v

This example would show the online help for -v option.

-A Default Session ID

Usage:

```
-A
```

Description:

Use the -A switch to override the default session ID in data signatures with the specified 16-bit ID.

Default:

Not set

-U I/O Signature Timestamp Units

Usage:

```
-U
```

Description:

Use the -U switch to set timestamps in I/O signature in seconds (-U) or in milliseconds (-Um).

Default:

Not set

--steady-state Determine Steady State

Usage:

```
--steady-state=[r.][.tag]<iops|mbps|lat>[:<%range_deviation,%slope_deviation>]
```

Description:

Determine steady state per target across a measurement window of 5 runs.

Note: When all targets have reached steady state without error, pain/maim exits with exit code 100 which can be monitored using a driving script.

The options are:

r	If specified, start a new measurement window. If not specified, the measurement window continues from previous runs.
iops	Track IOPS for steady state.
mbps	Track MBPS for steady state.
lat	Track I/O latency for steady state.
tag	An arbitrary string that is not 'r', 'iops', 'mbps' or 'lat' which can be used to uniquely identify a steady state testing case. This tag will be added to the steady-state.csv file name.
%<range_deviation>	Allowed deviation of minimum and maximum tracked values from the average. Default, if not specified, is 20%.
%<slope_deviation>	Allowed deviation of minimum and maximum points in a best linear fit line through the tracked values. Default, if not specified, is 10%.

Examples:

```
pain --steady-state=r.iops:20,10 (note the 'r' to reset).
```

```
pain --steady-state=iops:20,10 (no reset)
```

```
pain --steady-state=mbps.MySSDTest:20,10
```
(Note that "MySSDTest" used as a tag.)

The following is a sample test script written as a Windows batch file:

```
@echo off
SetLocal EnableDelayedExpansion
pain -f\\.\physicaldrive1 --full-device -b128k -l35 -u -w -i2 -t8
echo Running initial loop.
pain -f\\.\physicaldrive1 -b4k -w –d60 --steady-state=r.iops -Y1
echo Starting steady state loop.
FOR /L %%l IN (2,1,25) DO (
pain -f\\.\physicaldrive1 -b4k -w –d60 -m17 --steady-state=iops -Y1
IF !ERRORLEVEL! EQU 100 goto DONE
)
goto EXIT
:DONE
echo Steady state reached.
:EXIT
```

Default:

Not set

--latency-histogram Collect Latency Histogram

Usage:

```
--latency-histogram=<upperbound1[,upper_bound2[,...]]>
```

Description:

Collect latency histogram per target.

The collection bins are specified using a comma-separated list specifying the upper bound of each bin. The list is sorted by the magnitude of the upper bound values, and the range of each bin is constructed such that the upper bound is as specified and the lower bound is the upper bound of the previous bin.

Upper bounds may be specified as a floating point value (e.g. "0.5" or "4.5").

The time unit suffix may be used:

'n' for "nano" 'u' for "micro" 'm' for "milli" 's' for "seconds"

If no time unit suffix is given, 'm' for "milli" is assumed as a default.

Examples:

```
--latency-histogram=100u,500u,1m,3,5,10
```

The "Bin" column lists the upper-bound of the range as you give it in the command line. The "Upper (msec)" column is the upper bound value normalized to milliseconds. The other columns, R%, W%, and R+W% display the percentage of Reads Write, or Read/Write operations with measured latency that are within the bin; while CR%, CW%, and CR+W% display the cumulative value of the percentage for Read, Write, or Read/Write operations with measured latency through each bin. The last row, rest, is a bin that is added for operations with latency greater than the largest specified bin. INF (for infinity) is inserted in this row as this bin cannot be normalized.

```
LATENCY: TARGET:1 (\\.\PhysicalDrive1)
    Bin,Upper (msec),      R%,     CR%,      W%,     CW%,    R+W%,    CR+W%
   100u,          0.1,    21.4,    21.4,    21.2,    21.2,    21.3,    21.3
   500u,          0.5,    74.5,    95.9,    74.5,    95.7,    74.5,    95.8
     1m,            1,    3.32,    99.2,    3.61,    99.3,    3.47,    99.2
      3,            3,   0.789,     100,   0.737,     100,   0.763,     100
      5,            5,       0,     100,       0,     100,       0,     100
     10,           10,       0,     100,       0,     100,       0,     100
   rest,         +INF,       0,     100,       0,     100,       0,     100
```

Default:

Not set

Stand-alone Switches

The switches described in this section are the stand-alone switch commands.

-Z License Client Operation

Usage:

```
-Z
```

Description:

This switch is used for license operations. To checkout a license, use `-Z` with the desired number of days (ex. `-Z3`). To check-in a license, use a day value of `0` or `-Z` by itself (ex. `-Z0` or `-Z`). This switch is also used to activate a remote license. The syntax is `-Z#license_file_name.lic`, where `license_file_name.lic` is the name and path to a remotely checked out license file.

Default:

Not set

--secure-erase Erase the Target Device and Exit

Usage:

```
--secure-erase[=[ata_command][,[scsi_command]][t]]
```

Description:

Erase the target device and exit.

By default, this option sends a SECURITY ERASE command to ATA devices, a SANITIZE BLOCK ERASE command to SCSI devices, and a FORMAT UNIT command to NVMe devices (through SCSI translation layer). FORMAT UNIT is the only available command for NVMe devices. For ATA or SCSI devices, optional "ata_command" or "scsi_command" variations may be specified as follows.

The "`ata_command`" may be specified immediately after '`=`' and can be one of the following:

1 SECURITY ERASE

2 ENHANCED SECURITY ERASE

The "scsi_command" may be specified after ',' and can be one of the following:

1 SANITIZE BLOCK ERASE

2 SANITIZE CRYPTOGRAPHIC ERASE

3 SANITIZE OVERWRITE (INVERT=0, OVERWRITE COUNT=1)

4 SANITIZE OVERWRITE (INVERT=1, OVERWRITE COUNT=1)

5 SANITIZE OVERWRITE (INVERT=1, OVERWRITE COUNT=2)

6 FORMAT UNIT

If either "ata_command" or "scsi_command" is omitted, the command value defaults to '1'. For example:

"--secure-erase" is the same as "--secure-erase=1,1"

"--secure-erase=2" is the same as "--secure-erase=2,1"

"--secure-erase=,5" is the same as "--secure-erase=1,5"

The optional 't' suffix enables a fall-back to full-device TRIM command for ATA and UNMAP for SCSI and NVMe devices if the specified erase command fails. For example, if the SECURITY ERASE command fails on a SATA SSD because it is in SECURITY FROZEN state, the 't' modifier sends a TRIM command for every LBA as a fall-back erase method.

This option requires valid targets to be specified. An ATA device must be in security state "SEC1" where the SECURITY features must be supported but not enabled or in some locked state. (Security states are defined in the INCITS ATA/ATAPI Command Set 3 documentation - refer to http://www.t13.org.) For example, it is not possible to perform SECURITY ERASE if the device is in SECURITY FROZEN or SECURITY LOCKED state.

The '-M' option may be used to specify the maximum wait time. When the wait time is exceeded, the test will attempt to exit immediately regardless of the status of the operation. If '-M' is not specified with '--secure-erase', then '-M0' is assumed for no time limit and the test will run until the operation completes successfully or returns an error.

Default:

Not set (i.e. normal I/O mode)

--trim Send Trim to Target

Usage:

```
--trim
```

Description:

The Trim command allows an operating system to inform a solid-state drive (SSD) which blocks of data are no longer considered in use and can be wiped internally. Send TRIM to target device and exit.

Pain/Maim use the following optional parameters to determine the LBA ranges for TRIM: '-t', '-Q', '-b, '-O', '-x', '-m', '--full-device', and file size (with or without '--file-size').

The '-M' option may be used to specify the maximum wait time. When the wait time is exceeded, the test will attempt to exit immediately regardless of the status of the operation. If '-M' is not specified with '--trim', then '-M0' is assumed for no time limit and the test will run until the operation completes successfully or returns an error.

Default:

Not set

I/O Characteristic Switches

The switches described in this section are the I/O characteristic commands.

-b Buffer Size

Usage:

`-bbuffer_size[units]`

Description:

Use `-b` to specify the buffer size for each I/O. This equates to the I/O block size, from the application level. You can specify the buffer size in bytes or use a numeric value and unit designator:

`m` = megabytes

`k` = kilobytes

`b` = bytes (default)

u = LB (logical block) units, usually 512 bytes

Note: For physical drive targets, buffer size must be a multiple of device's logical block size (usually 512 bytes.) For other types of targets (such as a regular file), the buffer size must be a multiple of 512 bytes.

Examples:

`-b1m` = 1 megabyte (1048576 bytes.)

`-b4k` = 4 kilobytes (4096 bytes.)

`-b8192` = 8 kilobytes

Default:

The default buffer size in MLTT is 64 kilobytes (65536 bytes).

-B Sequential I/O Direction Control

Usage:

`-Bmode_number`

Description:

Use `-B` to control the "direction" of I/O traffic on a target. By default, I/Os will start at the beginning (lowest LBA) of the specified file or device offset, run to the specified file size (highest LBA), then repeat from the beginning. This switch allows you to request "backward" I/O runs (start at end of the file, highest LBA, or highest device offset and run to the beginning, lowest LBA, of the file).These modes are useful for video editing simulations. The available modes are:

0 = All I/O forward

1 = Forward / Backward / Forward, etc.

2 = First I/O Forward, Rest Backward

3 = All I/O backward

Default:

By default, all I/Os only run in the forward direction or `-B0`.

-c Commit or Flush Data

Usage:

`-c`

Description:

Use -c to specify that the tool should explicitly request a commit or flush of each write command. This switch is currently only supported when running to file system targets. This switch works independently of the write cache options (-W).

Default:

This option is disabled by default.

-g Burst Mode Interval

Usage:

```
-gseconds
```

Description:

This switch is used to set a time interval between I/O bursts in Maim. A burst equates to a dispatch of simultaneous requests corresponding to the queue setting (-Q). The time interval may be set in seconds or milliseconds. A numeric value by itself indicates seconds (ex. -g1 equals 1 second between bursts.) If 'm' is added to the number, milliseconds will be indicated. For example, -g10m equals 10 milliseconds between bursts. This switch only applies to the non-continuous queuing Maim I/O modes.

This option is ignored in pain or in maim when running in non-burst (continuous) queuing mode.

Default:

Burst mode is disabled by default and I/O groups are sent immediately, one after another.

-m I/O Call Method Mode Number

Usage:

```
-mmode_number
```

Description:

This option controls the following I/O traits.

Asynchronous I/O queuing mode (maim only):

- Burst queuing
  The requested queue-depth of I/Os (see -Q) are issued at once. The Test Tool waits for all pending I/Os to complete before issuing the next burst of I/Os.
- Continuous queuing
  The request queue-depth of I/Os are issued once initially. Rather than waiting for all I/Os to complete, the Test Tool issues a new I/O request each time one of the pending I/Os completes. Continuous queuing is used for maximizing the number of I/O requests per second (IOPS).

Device Access Mode:

- Sequential access
  The Test Tool issues each new I/O to a device offset that is sequentially adjacent to the previous I/O request.
- Random access
  The Test Tool issues each new I/O to a randomly chosen device offset.

Device Coverage Mode:

- Partial Coverage
  The Test Tool covers the area specified by file size.
- Full coverage
  The Test Tool covers the entire target device.

Memory Stream Copy:

By bypassing the actual I/O to a target device or file, the Test Tool effectively runs in memory-to-memory copy mode. This is a good way to test the system bus performance at different levels (i.e. L1 cache, L2/L3 caches, RAM). NOTE: MLTT cannot bypass the host operating system's virtual memory sub-system.

The **mode_number** may be one of the following:

1 General I/O mode

- Pain – synchronous I/O, sequential access
- Maim – asynchronous I/O, continuous queuing, strict sequential access

9 Memory stream copy (no I/O to device)

11 General burst queuing asynchronous I/O mode

- Pain – not applicable (reverts to -m1)
- Maim – asynchronous I/O, burst queuing, sequential access

16 Continuous queuing asynchronous I/O mode

- Pain – not applicable (reverts to -m1)
- Maim – asynchronous, continuous queuing, sequential access

17 Random access, full device coverage mode

- Pain – synchronous I/O, random access, full device coverage
- Maim – asynchronous I/O, continuous queuing, random access, full device coverage

18 Sequential access, full device coverage mode

- Pain – synchronous I/O, sequential access, full device coverage
- Maim – asynchronous I/O, burst queuing, sequential access, full device coverage

30 TCP/IP network I/O mode

Default:

The default mode is 1 for pain, 11 for maim, and 30 for sock.

-Q Queue Depth (Maim only)

Usage:

`-Qqueue_depth`

Description:

Use -Q to specify the maximum number of outstanding I/Os (queue depth) per worker thread in Maim, our asynchronous tool. The maximum number of outstanding (pending) I/Os is dependent on the operating system and memory resources. The practical limitation will also depend on the target capabilities.

This switch is ignored in Pain.

Default:

The default queue depth is one.

-r Read-only Mode

Usage:

`-r`

Description:

Use `-r` to indicate a read-only mode.

Important: By default, the tools will still do a single write FOP in order to lay down the specified data pattern.

After the write pass, the tools issue repeated read-backs of the data with data integrity checking enabled by default. If no initial write pass is desired, such as in performance tests where data is not relevant, you can combine the -r switch with the -n and -o switches for a pure read-only mode. This combination will result in the reads returning whatever data exists in the file or device area. You must specify an existing file, logical device, or physical device with a minimum size equal to or greater than the specified file size in order to perform reads only.

Example:

```
pain -f\\.\physicaldrive2 -r -n -o
```

Default:

By default, the tools perform both write and read operations, with data comparison.

-ro Read-only with One Write Pass

Usage:

`-ro`

Description:

This switch is a macro that is the same as specifying the `-n -r -o` switches, except that one write pass is performed. This is useful for cases where you do not want data comparisons, but you need a particular data pattern for read-only traffic. You might want to use this macro in signal integrity testing, with an in-line analyzer set to trigger on error conditions. The device or file is held open for the duration of the test to increase performance.

Default:

By default, the tools perform both write and read operations, with data comparison.

-R Read Buffering Mode

Usage:

`-R<0|1|2|3|4>`

Description:

The `-R` switch is used to set the file open/creation flags that can affect read buffering. The available read buffering modes are:

Windows	0 =	Do not explicitly set file open flags
	1 =	Cache allowed, no O/S buffering
	2 =	Cache allowed, O/S buffered
	3 =	Non-cached, no O/S buffering
	4 =	Non-cached, O/S buffered
Linux	0 =	Do not explicitly set file open flags
	1 =	O_DIRECT on, O_SYNC on
	2 =	O_DIRECT on, O_SYNC off
	3 =	O_DIRECT off, O_SYNC on
	4 =	O_DIRECT off, O_SYNC off

Solaris	0 =	Do not explicitly set file open flags
	1 =	O_SYNC on
	2 =	O_SYNC off
	3 =	O_SYNC on
	4 =	O_SYNC off

Default:

The default is 1, Cache allowed, No O/S buffering or `-R1`.

-s Single Sector I/O Mode

Usage:

`-ssectors`

Description:

Use `-s` to set the tool to a sector-based I/O mode. Basically, this switch acts as a macro to optimize for intense reading to a small area on a disk. Read I/O is contained to the number of sectors specified. The following flags are set automatically: `-q1 -r -o -n -R3`, except on non-Windows platforms, the `-R3` flag is not set automatically. When you use this switch, you must specify the file/device by using the `-f` switch. This switch is not available in all tools. Refer to the command line help.

Default:

This option is disabled by default.

-t Thread Count

Usage:

```
-tthread_count [p]
```

Description:

Use -t to specify number of worker threads in pain, maim, and sock. In sock, this corresponds to the number of concurrent socket connections per sock process. In pain (synchronous I/O), since each thread dispatches a single I/O at a time, this number roughly correlates to queue depth. In maim (asynchronous I/O), each thread ties to maintain concurrent I/O operations specified by -Q. Therefore, in maim, the potential maximum I/O operations in-flight per target device is the product of the thread count and the queue depth (Thread Count X Queue Depth). For examples, refer to Figure 56 on page 132.

Figure 57 on page 133 shows the [p] option with the <thread_count> command. With 'p' used, <thread_count> specifies the maximum number of total I/O threads to use per process. So rather than creating "threads-per-target", it assigns "targets-per-thread". This option is valid only for Maim.

Note: If you specify the special case "-t0p", then the program sets the <thread_count> to the number of available CPUs. For example, if the system has 4 CPUs, then "-t0p" is same as "-t4p". If in that same system you also specify "-T2" (i.e. use only the first 2 CPUs), then "-t0p" is same as "-t2p".

With the 'p' modifier, the total number of I/O threads is at most the specified <thread_count> value, no matter the target count.

Example:

The following switch specifies that 8 separate I/O generating threads will be created from the central application.

```
-t8
```

Default:

The default thread count is one.

-w Write-only Mode

Usage:

```
-w
```

Description:

The -w switch is used to indicate a write-only mode. No reads are performed. This option is ignored if -%r or -%w is specified.

Default:

By default, the tools perform both write and read operations, with data comparison.

-W Write Buffering Mode

Usage:

`-W<0|1|2|3|4>`

Description:

Use the `-W` switch to set file open/creation flags that can affect write buffering. The available write buffering modes are:

Windows	0 =	Do not explicitly set file open flags
	1 =	Cache Allowed, no O/S buffering
	2 =	Cache Allowed, O/S buffered
	3 =	Non-cached, No O/S buffering
	4 =	Non-cached, O/S buffered
Linux	0 =	Do not explicitly set file open flags
	1 =	O_DIRECT on, O_SYNC on
	2 =	O_DIRECT on, O_SYNC off
	3 =	O_DIRECT off, O_SYNC on
	4 =	O_DIRECT off, O_SYNC off
Solaris	0 =	Do not explicitly set file open flags
	1 =	O_SYNC on
	2 =	O_SYNC off
	3 =	O_SYNC on
	4 =	O_SYNC off

Default:

The default is `1`, Cache Allowed, No O/S Buffering or `-W1`.

-% I/O Profile Specification

Usage:

`-%r<weight>[@buffer_size]` for read percentage

`-%w<weight>[@buffer_size]` for write percentage

`-%f<weight>` for sequential access forward percentage

`-%b<weight>` for sequential access backward percentage

`-%x<weight>` for random access percentage

`-%s<random_seed>` 32-bit seed for I/O profile random number generation

`-%o<size>` Random offset alignment size - insures that random I/Os are issued on boundaries of the indicated size.

`-%t<spec[,spec[,spec[,...]]]>` Set the I/O read/write and access mix profile with one or more of the following 'spec' keys.

Description:

The `-%` option can be used to specify a mix of operations and I/O access positioning. This is useful in generating I/O modeling real world applications.

Use `-%r` and `-%w` to specify what percentage of I/O operations should be reads or writes. For example, `-%r10 -%w90` states “10% read operations, 90% write operations.” The `<weight>` value is required for each `-%r` and `-%w` specification. The `[@buffer_size]` modifier is optional and can be used to specify the buffer size used for a `-%r` or `-%w` specification. Without the optional `[@buffer_size]` modifier, either the global buffer size specified by the `-b` option (if specified) or the default buffer size of 64KBs is used. The `-%r` and `-%w` specifications are accumulative and can be used multiple times in the command line. The final percentages are determined from the sum of all `<weight>` values for all `-%r` and `-%w` options given in the command line.

Use `-%f`, `-%b`, and `-%x` to specify the probability of I/O direction. `-%f` determines the probability of next I/O position to be sequentially forward adjacent from the previous I/O position. `-%b` determines the probability of next I/O position to be sequentially backward adjacent from the previous I/O position. `-%x` determines the probability of next I/O occurring at any random position within the target coverage area. The `<weight>` value is required for each `-%f`, `-%b`, and `-%x` specification. Only one instance of each of `-%f`, `-%b`, and `-%x` specifications is valid in the command line. The final I/O access position percentages are determined from the sum of all `<weight>` values for all `-%f`, `-%b`, and `-%x` options given in the command line.

Use `-%s` to specify a 32-bit number used as the seed for I/O profile random number generation. If `-%s` given more than once in the command line, the last instance takes effect.

Transaction mode:

- T[N]

Enable transaction simulation mode, with optional value 'N' denoting the number of transactions per connection. If 'N' is not given, the number of transactions per connection is unlimited. If 'N' is specified as a 'MIN-MAX' range, a new random value between 'MIN' and 'MAX' is used for each new connection.

Read/write operation mix:

- r<weight>[@<size>]

  Read operation probability 'weight' with optional buffer 'size'.

'Size' can be specified as 'MIN-MAX' range for random values. In transaction mode ('-%T' enabled), this specifies the transaction request size.

- w<weight>[@<size>]

  Write operation probability 'weight' with optional buffer 'size'.

'Size' can be specified as 'MIN-MAX' range of random values. In transaction mode ('-%T' enabled), this specifies the transaction response size.

Forward/backward/random access mix:

- f<weight>

  Forward sequential access probability 'weight'

- b<weight>

  Backward sequential access probability 'weight'

- x<weight>

  Random access probability 'weight'

- o<size>

  Random offset alignment 'size'

Random seed:

- s<seed>

32-bit seed for random number generators used for read/write mix and I/O access sequences

Default:

Unset

Examples:

-b8k -%r10 -%r10@4k -%w20 -%w15@512b -%w45@1k

In this example, since the sum of all weight values is 100, each weight value corresponds to the exact percentage. Looking at each option:

`-b8k`	specifies the default buffer size of 8KBs
`-%r10`	specifies “10% reads using the default 8KB buffer size”
`-%r10@4k`	specifies “10% reads using 4KB buffer size”
`-%w20`	specifies “20% writes using the default 8KB buffer size”
`-%w15@512b`	specifies “15% writes using 512-byte buffer size”
`-%w45@1k`	specifies “45% writes using 1KB buffer size”

In the next example, the sum of all weight values is not 100; therefore, the percentages are calculated relative to the actual sum.

`-r10 -w30`	using the final sum of 40, this specifies “25% reads, 75% writes”

The following example demonstrates how to specify the I/O access position mix:

`-%f30 -%b70`	specifies “30% forward sequential, 70% backward sequential.”
`-%f50 -%b70 -%f30`	also specifies “30% forward sequential, 70% backward sequential.” Because only one each of -%f, -%b, and -%x takes effect, the last -%f30 overrides the first -%f50.

In the following example, because the sum of the weight values is not 100, the percentages are determined relative to the actual sum:

`-%f40 -%b100 -%x60`	using the final sum of 200, this specifies “20% forward sequential, 50% backward sequential, 30% random access.”

In the following TCP/IP application transaction mode example, all requests are between 30 to 500 bytes in size, while 20% of responses are 4KBs, another 20% are 8KBs, and the remaining 60% or responses are 100KBs in size. The transactions-per-connection (or an application session) is between 20 to 30 transactions, after which each I/O thread terminates the connection and establishes a new one before continuing.

```
sock -%T20-30 -%r100@30b-500b,w20@4k,w20@8k,w60@100k
```

--scsi Direct SCSI Command for Read/Write

Usage:

```
--scsi[=0|1|2|3|4]
```

Description:

Use direct SCSI commands for read/write. This option is ignored if not running synchronous I/O to physical drive targets. The available mode numbers are:

0 = Off (default)

1 = On (READ10/WRITE10)

2 = On (READ10/WRITE10), with Forced Unit Access (FUA)

3 = On (READ16/WRITE16)

4 = On (READ16/WRITE16), with FUA

If '--scsi' is specified without using '0', '1', '2', '3', or '4', then '`--scsi=1`' is assumed.

Default:

```
--scsi=0
```

--skip Sequential I/O Skip Size

Usage:

```
--skip=<size>
```

Description:

In sequential I/O modes, use the --skip option to skip a specified amount of “<size>” between adjacent I/Os. This may be useful in cases when the operating system coalesces adjacent I/Os which can skew the observed IOPS count. The “<size>” parameter specification is the same as for buffer size (“-b<size>”).

Example:

Perform sequential I/O from LBA 0x100000 to LBA 0x900000 (i.e. file size of '0x900000 - 0x100000' or '0x800000' LBA units), 7 LBA units at a time, skipping 1 LBA unit between each I/O:

```
pain -x0x100000u -b7u 0x800000u --skip=1u
```

Default:

'`--skip=0`'

--cap Limit I/O Throughput

Usage:

```
--cap=[p|P|t|T][.]<limit>
```

Description:

Try to limit the I/O throughput to 'limit' bytes-per-second. The optional 'p' or 'P' prefix specifies the scope of the limit to be the process. The optional 't' or 'T' prefix specifies the scope of the limit to be per-target. Without the scope prefix, the limit is per-thread. Use the optional '.' separator to specify 'bits-per-second' rather than the default 'bytes-per-second'. An optional size unit suffix may be used (e.g. just as for file size and buffer size parameters.) If no unit suffix is given, the default unit of 'm' is assumed for 'megabytes-per-second' or, if '.' is specified, 'megabits-per-second'.

This option does not cap the actual device speed. It only tries to limit the I/O submission rate in order to sustain the specified limit over time.

Examples:

Cap the process throughput to 50 megabytes-per-second: `pain --cap=p50`

Cap the per-target throughput to 50 megabytes-per-second: `pain --cap=t50`

Cap the per-thread throughput to 50 megabytes-per-second: `pain --cap=50`

Cap the per-target throughput to 1 gigabits-per-second: `sock --cap=t.1g`

Cap the per-thread throughput to 1 gigabits-per-second: `sock --cap=.1g`

Default:

The default limit is '0' (no cap).

--perf-mode Performance-optimized mode

Usage:

```
--perf-mode
```

Description:

Use this option to run in performance-only testing mode to achieve the best I/O throughput. In this mode, the buffer usage is performance-optimized, and data integrity testing is not possible. This option automatically enables '`-n`', '`-N`', '`-u`', and '`-o`' options.

Target Related Switches

The switches described in this section are the target related commands.

File Size

Usage:

`file_size`

Description:

The desired "file" size is specified as a number, with no preceding switch argument. You can specify the size in bytes or use a numeric value and unit designator:

`g` = gigabytes

`m` = megabytes (default)

`k` = kilobytes

u = LB (logical block) size units, usually 512 bytes

Examples:

```
1g
100m
512k
```

The file size must be at least the same as the I/O size, or a multiple of the I/O size.

Maximum file size is as allowed by the operating system.

The file size can apply to an actual file in file system based testing or the extent of linear space to utilize on a logical or physical drive. Note that file size will be utilized per thread (that is, each worker thread in our thread-based tools will utilize the extent specified by the file size – 8 threads x 100m [`-t8 100`] would equal 800 megabytes total.) The default file size varies in each tool.

Default:

The default file size per thread in Pain is 4MB.

The default file size for the single worker thread in Maim is 10MB.

-f Target

Usage:

`-ftarget`

Description:

Use `-f` to specify the desired target. The target can be a file, logical drive, or physical drive that resides in the host system or is externally attached via SCSI, USB, FireWire, LAN, SAN, and others.

Also, when using sock, the target may specify the hostname or an IP (or IPv6) address of a peer for TCP/IP network I/O.

Note: If the switch is not specified, one file of the specified size is created in the current directory by each worker thread.

Default:

If no target is specified, each worker thread creates a file in the current directory.

Examples:

Physical: `-f\\.\physicaldrive1`

Logical: `-f\\.\g:`

File: `-fg:\file1.dat`

Linux device: `-f/dev/sdc`

Solaris device: `-f/dev/rdsk/c1t1d0s2`

TCP/IP peer (sock): `-fhostname or -f10.23.1.101 or -f10.23.1.80:10.23.1.101` (where `10.23.1.80` is a specific local IP if there is more than one network interface to choose from). When specifying an IPv6 address pair, use '-' to separate the local and remote addresses (e.g. `-ffe80::c62c:3ff:fe08:a66c%en0-fe80::221:9bff:fe50:90ec`). For a link-local IPv6 address pair, the scope ID of the outgoing interface must be appended to the local address (e.g. `"%en0"` or `"%5"`).

The tools also support a multi-target mode, where multiple targets can be accessed in a single process. The Catapult -t switch option performs this automatically. See "-t Multi-target mode" on page 226. Multiple targets may also be specified manually in one of several manners:

- Create a text file called "targets.dat" that contains desired targets, one per line. Catapult can create this file for you. For example:

  ```
  catapult -p -t
  ```

  Then pass this file name, with path if necessary, to pain or maim with the -f switch.

  ```
  pain -ftargets.dat
  ```

- You can also specify multiple targets on the command line, separated by commas. For example:

  ```
  pain -f\\.\physicaldrive1,\\.\physicaldrive2
  ```

- You can also use a prefix system, where a common prefix is terminated with a semi-colon, followed by suffixes that are comma separated. For example:

```
pain -f\\.\physicaldrive;1,2,3
```

- Generate TCP/IP I/O to `10.23.1.101` and `10.23.1.102` from `10.23.1.80`.

```
sock -f"10.23.1.80:10.23.1.;101,102"
```

NOTE: on Unix systems, the shell interprets ';' as the command separation character; therefore, the target name should be quoted. For example, the shell interprets the following:

```
pain -f/dev/sd;b,c,d
```

as a sequence of two commands as shown below:

```
pain -f/dev/sd
b,c,d
```

To prevent such errors, the target specification must be quoted.

```
pain -f"/dev/sd;b,c,d"
```

Warning: Physical and logical drive access is destructive! Existing data WILL be overwritten.

-o Keep Target Device or File Open

Usage:

```
-o
```

Description:

Use `-o` to disable repeated opening and closing of the file or device. This switch causes the file or device to be opened once and kept open for the duration of the test. Keeping a file or device open increases performance.

The `-o` option is implied for continuous queuing maim modes or if random access is specified (for example, through a combination of `-%f`, `-%b`, `-%x`).

Default:

By default, the tools open and close the file or device with each FOP.

-O Override Device Base Offset

Usage:

```
-O[mode]
```

Description:

Use -O to override the MLTT default device base offset setting. By default, I/O starts at a 1MB offset on the specified device. This switch instructs the tools to start I/Os at the start of the device (that is, sector 0 on a hard drive). You should always use this switch when running I/O to an existing file on a file system partition to avoid unnecessary seeking. For example:

```
-fg:\test.dat -O
```

Caution: Use this switch with extreme caution on physical drives or logical partitions! Overwriting a drive from sector 0 will erase OS-specific details, such as the drive signature.

Default:

By default, I/O starts at a 1MB offset on the specified device.

-x Multi-Share Mode 1 - Multiple Sessions Offset

Usage:

`-xoffset`

Description:

Use `-x` to enable Multi-Share Mode 1. This mode allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently. You can optionally specify the starting offset number in megabytes. A number supplied with this switch will be used to set the base (starting) offset for the file/device in megabytes.

To avoid collisions with other sessions of the tools, you must set the base offset for each new session beyond the highest offsets of previous sessions. Offset maybe specified with an optional unit: 'b' for bytes, 'k' for kilobytes, 'm' for megabytes, 'g' for gigabytes. The offset value must be a multiple of the logical block size of the target device.

Example:

machine a: `pain -t10 10 -x1` (runs 10 threads at 10MB each starting at 1MB offset)

machine b: `pain -t10 10 -x101` (runs 10 threads at 10MB each starting at 101MB offset)

In this example, the base offset for machine b is set to the lowest possible value that will not conflict with machine a. (i.e. 10 threads multiplied by a per-thread file size of 10 equals 100MB of space used by machine a. Machine b has to start at a minimum offset of 101MB to avoid overwriting machine a.)

Default:

The default offset, if `-x` is specified without the optional offset value, is 0MB if `-O`, `-O1`, `-O2`, or `-O3` is specified; otherwise, the default offset is 1MB.

-X Multi-Share Mode 2 - All Threads Issue I/Os to the Same Offsets

Usage:

```
-Xoffset
```

Description:

Use the -X switch to enable Multi-Share Mode 2. In this mode, all threads issue I/Os to the same offsets. The offset maybe specified with an optional unit: 'b' for bytes, 'k' for kilobytes, 'm' for megabytes, 'g' for gigabytes. The offset value must be a multiple of the logical block size of the target device. A number supplied with this switch will be used to set the starting offset to use for the file/device in megabytes. This mode automatically disables data pattern reversals and unique I/O marks to prevent false data corruptions.

Default:

The default offset, if `-x` is specified without the optional offset value, is 0MB if `-O`, `-O1`, `-O2`, or `-O3` is specified; otherwise, the default offset is 1MB.

--full-device Run to Entire Target Device

Usage:

```
--full-device
```

Description:

This is a convenience option to specify full-device coverage if the target is a physical device or a volume. The testing area size is determined by the device size, starting offset, and the thread count - i.e. Per-thread testing area is (device size - starting offset) / thread count.

The transfer size of the very last I/O to the end of the device may be smaller than the buffer size specified by “-b”.

Default:

Enabled for -m17 and -m18, disabled for all other -m modes.

--smart S.M.A.R.T Monitoring

Usage:

```
--smart -f<targets> [other options]
```

Retrieves Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T. - also written as SMART) attributes and status from target devices and logs them.

At the end of the test, the overall SMART status is output to screen for each target. For example:

```
SMART: Target 1: '\\.\physicaldrive1' - OK
```

In addition, the .log file contains the retrieved SMART attributes and data in CSV format. The following is the output of an example .log file using the SMART command:

```
SMART: Target 1: '\\.\physicaldrive1' - OK
    ATTR (HEX), VALUE,         FLAG BITS (HEX),    VENDOR SPECIFIC DATA, DESC (from Wikipedia)
         5 (05),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, Reallocated Sectors Count
         9 (09),   100, 0000000000110010 (0032), 64 C3 25 00 00 00 00 00, Power-On Hours (POH)
        12 (0C),   100, 0000000000110010 (0032), 64 D3 02 00 00 00 00 00, Power Cycle Count
       170 (AA),   100, 0000000000110011 (0033), 64 00 00 00 00 00 00 00, Available Reserved Space
       171 (AB),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, SSD Program Fail Count
       172 (AC),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, SSD Erase Fail Count
       174 (AE),   100, 0000000000110010 (0032), 64 C7 02 00 00 00 00 00, Unexpected power loss count
       175 (AF),   100, 0000000000110011 (0033), 64 F7 02 B6 0E 2C 00 00, Power Loss Protection Failure
       183 (B7),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, SATA Downshift Error Count or Runtime Bad Block
       184 (B8),   100, 0000000000110011 (0033), 64 00 00 00 00 00 00 00, End-to-End error / IOEDC
       187 (BB),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, Reported Uncorrectable Errors
       190 (BE),    75, 0000000000100010 (0022), 40 19 00 15 2C 00 00 00, Airflow Temperature
       192 (C0),   100, 0000000000110010 (0032), 64 C7 02 00 00 00 00 00, Power-off Retract Count
       194 (C2),   100, 0000000000100010 (0022), 64 19 00 00 00 00 00 00, Temperature resp
       197 (C5),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, Current Pending Sector Count
       199 (C7),   100, 0000000000111110 (003E), 64 00 00 00 00 00 00 00, UltraDMA CRC Error Count
       225 (E1),   100, 0000000000110010 (0032), 64 CD 21 8C 00 00 00 00, Load/Unload Cycle Count
       226 (E2),   100, 0000000000110010 (0032), 64 FF FF 00 00 00 00 00, Load 'In'-time
       227 (E3),   100, 0000000000110010 (0032), 64 FF FF FF FF 00 00 00, Torque Amplification Count
       228 (E4),   100, 0000000000110010 (0032), 64 FF FF 00 00 00 00 00, Power-Off Retract Cycle
       232 (E8),   100, 0000000000110011 (0033), 64 00 00 00 00 00 00 00, Available Reserved Space
       233 (E9),    97, 0000000000110010 (0032), 61 00 00 00 00 00 00 00, Media Wearout Indicator
       234 (EA),   100, 0000000000110010 (0032), 64 00 00 00 00 00 00 00, Average erase count AND Maximum Erase Count
       241 (F1),   100, 0000000000110010 (0032), 64 CD 21 8C 00 00 00 00, Total LBAs Written
       242 (F2),   100, 0000000000110010 (0032), 64 13 0B AE 00 00 00 00, Total LBAs Read
```

The five column headings in the CSV file are:

ATTR (HEX) – is the SMART attribute ID (the hexadecimal equivalent in shown in parentheses).

VALUE – is the current raw decimal value of the attribute. Whether or not the condition represented by the attribute ID is in good or bad shape depends on whether or not this value is within the vendor-specific threshold value. There is no standard way to retrieve the threshold value itself as MLTT does not have access to the threshold value.

FLAG BITS (HEX) – is the 16-bit flags for the attribute retrieved from the drive (the hexadecimal equivalent in shown in parentheses).

VENDOR SPECIFC DATA – is the 8-bytes of vendor-specific data for the attribute retrieved from the drive and displayed in hexadecimal.

DESC (from Wikipedia) – These descriptions are not defined by the standard, and many attribute IDs are vendor-specific.

Data Pattern Related Switches

This switch category contains switches that you use when you specify patterns for your test.

For more information about data patterns refer to Chapter 6, "Data Pattern Reference."

-D Display the Data Pattern

Usage:

`-Dbytes`

Description:

Use `-D` for a visual representation of the selected data pattern on the console. Specifying this switch alone or with a numeric byte value causes the data pattern to run on the console in binary format. No I/Os are sent to any other device. The byte value is used to indicate the number of bytes wide that the pattern should take up on the console. Note that because the data pattern representation is in binary, each byte indicated will take up 8 character places on a console line. This switch is extremely useful for understanding the signal transitions induced by a particular data pattern. In addition to real time data pattern display, this switch can be used for a quick data pattern preview.

If an asterisk (`-D*`) is used instead of a numeric byte value, a brief excerpt of the selected data pattern will be displayed on screen in hex format, with no I/O to any devices. Note that this feature is not yet available in all tools. This feature is useful for validating that the data pattern characteristics specified on the tool command line are as expected. If `-D*b` is specified, the pattern preview is displayed in binary format instead of the default hex format. For `-D*` and `-D*b`, use `-b` (buffer size) option to specify the number of bytes to preview.

Default:

Using `-D` without a specified byte value will run the data pattern at a length of 8 bytes per line displayed on the console (the line across the console screen—before carriage return to the next line).

-e Custom Blink Pattern Modifier

Usage:

`-ebit_length`

Description:

The `-e` switch is a modification option for the custom blink pattern variations (`-l99`). The desired length of blinking (on) bits is specified with the bit length option. This switch can be used in conjunction with the `-L` switch to create some interesting patterns across various bus lengths. There is no default value and a bit length value must be specified. Refer to "-e Custom Blink Pattern Modifier" on page 199 for a detailed usage example.

Default:

There is no default value.

-E Custom Blink Pattern Modifier (for walking bit variations)

Usage:

`-Ehold_cycles`

Description:

The `-E` switch is a modification option for the custom blink pattern with walking bit options (`-l99w, -l99o, -l99f`). Hold cycles indicates the number of times that a pattern is repeated before the bit is walked. This switch can be useful in testing for stuck bit faults on bus architectures. There is no default value; you must specify a hold cycle value. Refer to "-E Custom Blink Pattern Modifier (for walking bit variations)" on page 200 for a detailed usage example.

Default:

There is no default value.

-F Custom Blink Pattern Modifier

Usage:

```
-F
```

Description:

The `-F` switch is a modification option for the custom blink pattern variations (`-l99`). This switch causes a “flip/flop” transition in the pattern by returning to the initial pattern value in each cycle. This switch is useful in testing for stuck bits. Refer to “-F Custom Blink Pattern Modifier” on page 201 for a detailed usage example.

Default:

This option is disabled by default.

-I Invert Pattern Mode

Usage:

```
-I
```

Description:

The `-I` switch is a modification option for certain data patterns. This switch causes a bit inversion of the data pattern with each transition cycle. Refer to the MLTT command-line help for a listing of data patterns that support this option. This switch is used to create some interesting bit-blink variations over bus architectures. Refer to “-I Invert Pattern Mode” on page 196 for a detailed usage example of this switch.

Default:

This option is disabled by default.

-l Specify a Data Pattern Number

Usage:

`-l`*`pattern_number`*

Description:

Use `-l` to specify the desired data pattern number. Most likely, you would want to indicate a specific data pattern for any test involving data or signal integrity. Refer to Chapter 6, “Data Pattern Reference” for more details about using this switch and other related switches.

Default:

The default pattern varies in MLTT.

-j Data Scrambling Mode

Usage:

`-jnumber`

Description:

Use -j to enable pre-scrambling of data patterns. Specify the desired scrambling mode. If -j is not specified, it is assumed that there is no prescrambling or -j0.

0 = No prescrambling

1 = SAS scrambler (reset scrambler every 1024 bytes of data)

2 = SATA scrambler (reset scrambler every 8192 bytes of data)

Use with the -J switch to override the default reset interval.

Default:

This option is disabled by default.

-J Data Scrambling Mode Reset Interval

Usage:

`-Jbytes`

Description:

Use -J with the -j switch to specify the scrambler reset interval in bytes. For SAS/SATA scrambling, this number should correlate to the data payload size. Use this switch only when you want to override the default scrambler reset values used by the -j switch.

Example:

Specifying `pain -j2 -J2048` would run the SATA scramble method, overriding the default data length of 8192 with 2048.

-L Number of Times to Repeat the Data Pattern Cycle

Usage:

`-Lcycle_length`

Description:

Use `-L` to modify the length or repetition of the selected data pattern cycle. The cycle length indicates the number of times to repeat each cycle of a data pattern before moving to the next unit. In general, the unit of data pattern refers to its length in bytes or bits. Refer to the command-line help for a listing of data patterns that currently support this option.

Example:

Specifying `-L4` as a modifier causes an 8-bit pattern to run each unit (one byte) four times before moving to the next unit, effectively creating a 32-bit pattern. Specifying `-L2` causes a 64-bit pattern to run each unit (eight bytes) two times, effectively creating a 128-bit pattern. Refer to "-L Number of Times to Repeat the Data Pattern Cycle" on page 195 for a detailed usage example.

Default:

The default value is one (no effect on the data pattern.)

-N Disable Data Pattern Reversals

Usage:

`-N`

Description:

Use `-N` to disable data pattern reversals on those patterns that support reversals. By default, most data patterns reverse after each FOP (forward, then backward). In general, reversals should be allowed anytime data comparisons are being performed as a means of insuring that stale data is not being read. See "Continuously Changing I/O Stream" on page 192 for more information about data pattern reversals.

Default:

By default, most data patterns reverse after each FOP (forward, then backward).

-P Modify Data Patterns with a Phase Shift

Usage:

`-Pcycle_length`

Description:

Use `-P` to modify supported data patterns with a "phase shift." This switch works on most blinking data patterns. Refer to the command-line help for a listing of data patterns that currently support this option. The effect is one of shifting the data pattern "out of phase" at the specified cycle length, such that the square wave, created by the on/off bits in the blinking byte values, reverses. You can specify the frequency of this shift as the cycle length or a default value will be used if you only specify `-P`. A specified cycle length is multiplied with pattern length to determine the frequency of the shift.

Example:

`-P32` would cause a 64-bit pattern to run for 256 bytes before shifting. (64-bits equals 8 bytes. 8 times 32 equals 256 bytes.) Refer to "-P Modify Data Patterns with a Phase Shift" on page 197 for an in-depth discussion on the use of this switch and a detailed usage example.

Default:

The default length varies by pattern and is equivalent to the full cycle length of the pattern.

-y Create Data Patterns Based on Various Lengths

Usage:

`-ypattern_value`

Description:

Use `-y` to specify a value to repeat in the write buffers and create the data pattern. Use this switch with several data pattern numbers `(-l)` to create patterns based on various lengths. Refer to Appendix A, "Data Pattern Numbers," or the MLTT command-line help for a listing of data patterns that support this option. Refer to "-y Create Data Patterns Based on Various Lengths" on page 202 for more information on this switch. For data patterns involving random number generation, this option may be used to specify a 32-bit random number seed value.

Default:

A default value is assigned that is equivalent to the thread index number.

Data Integrity Related Switches

The switches described in this section are the target related commands.

This category of switches is used to change the options related to data integrity checking. By default, the tools implement several measures to ensure that data read compares exactly with data written.

-C Comparison Mode

Usage:

`-Ccompare_mode_number`

Description:

Use `-C` to specify the data comparison mode you want. In general, it is desirable to always do a byte-for-byte comparison of write and read data, in order to catch any possible data corruption. However, there may be cases (usually due to system limitations) where the overhead of full buffer comparisons has a negative effect on I/O throughput to the target. In these cases, you can set the compare mode to only perform a check on the unique I/O signature in the data buffer. This substantially reduces processor utilization in the host system. Refer to Chapter 6, "Data Pattern Reference" for further discussion of I/O signatures. The data comparison modes are:

0 =	Disable data comparison
1 =	Full byte-for-byte compare (default)
2 =	Compare signatures only (2-3 words every 512 bytes)
3 =	Compare session id only (16 bit id at 2nd word every sector, typically every 512 bytes)
4 =	Session id compare, followed by write/read/full compare. Use with -A switch with options 3 and 4 to scan for specific session id.

Note: To turn off data comparisons completely, use the `-n` switch. This has the same effect as using the "0" option.

Default:

By default, the tools will perform a full byte-for-byte comparison of data read against data written.

-n Disable Data Corruption Checking

Usage:

```
-n
```

Description:

Use the `-n` switch to disable data corruption checking (write and read buffer comparisons.) This switch is normally used for performance testing. Host system processor utilization is greatly decreased when data comparisons are disabled.

Default:

Data corruption checking is enabled by default.

-u Disable Unique I/O Marks

Usage:

```
-u
```

Description:

Use `-u` to disable the unique I/O signatures placed in the data buffers by the tools. You normally use this switch for performance testing. I/O signatures are enabled by default and occur every 512 bytes in the I/O buffer. The signatures are extremely useful for debugging I/O errors and catching corruptions due to stale data. Refer to Chapter 6, “Data Pattern Reference” for further discussion of I/O signatures. Host system processor utilization may be slightly decreased and I/O throughput may slightly increase when I/O signatures are disabled.

Default:

I/O signatures are enabled by default.

-V Reverify Existing Data to a Specified Data Pattern

Usage:

```
-V
```

Use `-V` to instruct the tools to reverify existing data to a specified data pattern. This switch assumes that the specified file or device contains a previously written copy of the specified data pattern. You must specify this switch along with the EXACT data pattern and I/O characteristics used to write the data pattern previously. Because of the unique characteristics built into the data patterns, the data must also have been written with the -i1 switch specified on the command line. Only a single read pass (FOP) will be performed on the file or device. This switch is normally used as a test of targets which cache large amounts of write data, in order to validate, at a later time, that the data was successfully committed to disk. You can use this switch to test data backup or snapshot implementations.

Example:

Write a data pattern (note that -i1 is specified):

```
pain -l99 -L16 -i1 -w 100 -fg:\test.dat
```

Read the pattern back from a backup location:

```
pain -l99 -L16 100 -V -fh:\test.dat
```

Default:

This option is disabled by default.

Error Related Switches

This section describes switches related to the tools' handling of error conditions during testing.

-H Time to Wait Before Retrying an I/O Operation

Usage:

`-Hseconds`

Description:

Use the `-H` switch to specify the number of seconds to wait before retrying an I/O operation that previously encountered a non-fatal error. Retries, when possible, occur immediately by default. This switch can be used in conjunction with the -v switch, described in "-v Verify/Retry Count" on page 180.

Default:

By default, retries are performed immediately.

-M I/O Monitoring Mode

Usage:

`-Mseconds`

Description:

Use the `-M` switch to enable the I/O monitoring mode. The tools will display a warning when I/Os are not completed before the specified number of seconds (for example, -M60 would display a warning after 60 seconds.) By default, warnings will appear when a completion exceeds the performance sample time (5 seconds is default sample time.) You can specify a desired timeout or disable monitoring by indicating a timeout of 0 (-M0.) The I/O monitoring feature will report both complete I/O halts and individual stuck I/Os. You can also use this mode to catch I/O disruptions on an analyzer. If you use this switch with the -! or -# switches, an I/O trigger is sent when a halt or stuck I/O is detected. Note that there is no guarantee that the trigger I/O will reach the analyzer, as the target may be in an unresponsive state.

Default:

By default, I/O monitoring is enabled and errors are reported after the default performance sample interval of 5 seconds.

-v Verify/Retry Count

Usage:

```
-vretry_count
```

Description:

Use the -v switch to retry failed operations and to specify the number of retry attempts. You can use this switch with the -H switch described earlier in "-H Time to Wait Before Retrying an I/O Operation" on page 179.

Default:

By default, there is no retry on I/O errors (i.e. -v0), but a mandatory 1 retry on data corruption.

-! *(or -#)* Enable Analyzer trigger writes

Usage:

```
-! (-#)
```

Description:

This is the trigger debug flag. It instructs the tools to send a write I/O to the target device on critical errors with the idea that an analyzer can be set to trigger on the write data. It also generates additional log files that are extremely useful in regards to debug and analysis. The data value to trigger on occurs in the first two words of the data frame.

Note: -# can be used as an alternative to the -! command to enable the trigger debug flag.

The options associated with this switch are:

`-!` Writes `0xCACACACA 0xCACACACA` for data corruption trigger and `0xCACACACA 0xDEADBEEF` for I/O error trigger.

`-!2` Writes `0xDEADDEAD 0xDEADDEAD` for data corruption trigger, and `0xDEADDEAD 0xDEADBEEF` for I/O error trigger.

`-!3` Exits the application immediately - no trigger written.

`-!4` Executes external application and arguments specified in the MedusaTools.cfg file, which is located in the configuration folder where MLTT is installed.

The application and arguments can be specified in the configuration file as follows:

EXTERNAL_CMD=executable;

EXTERNAL_ARGS=arguments;

Example:

EXTERNAL_CMD=c:\test\myapp.exe;

EXTERNAL_ARGS=arg1 arg2 arg3;

This option can be used to trigger the Xgig Analyzer to start (trigger) or stop capture. The application and arguments that must be specified in the configuration file MedusaTools.cfg. For example, to trigger the Analyzer operating in the domain “My Domain (1,1,1) XGIG01001234”, set the application and arguments as follows:

EXTERNAL_CMD=triggeranalyzer.cmd;

EXTERNAL_ARGS=“My Domain(1,1,1)” XGIG01001234;

Ensure that the batch files, TriggerAnalyzer.cmd and StopAnalyzer.cmd, and the executable, wget.exe, are in the executable path. The executable, wget.exe, can be downloaded from: http://users.ugent.be/~bpuype/wget/

Now, the trigger can be activated by using the !4 option. For example, to trigger the Analyzer to capture on activation of the Pain tool,

pain -!4

See also *Xgig Analyzer User Guide* Appendix D.

`-!5` Writes default (0xCACACACA) trigger and exits immediately.

Default:

Triggering is disabled by default.

--handler Specify Custom Error Handling

Usage:

```
--handler=<error[,error[,...]]>:[<spec>[,<spec>[,...]]]
```

Description:

Specify error handling. Use multiple times to handle a list of errors.

Handled 'error' values (case insensitive):

`- 'size'`	I/O transfer size mismatch
`- 'corrupt'`	Data corruption
`- 'read'`	I/O error during read (reported by the OS)
`- 'write'`	I/O error during write (reported by the OS)
`- 'open'`	OS error while opening the target
`- close'`	OS error while closing the target
`-'flush'`	OS error during flush (sync)
`- 'timeout'`	I/O pending for longer than monitor period (see '-M')
`-'halt'`	No I/O reported for the process during sampling interval
`-'all'`	All handled errors

Error handler specs:

`- l<i|w|e>`	Label it as info ('i'), warning ('w'), or error ('e') - by default all error events are labeled as 'error'. The other error handling specs are ignored unless the error event is labeled as 'error'
`- t<n|r|x|b>`	No trigger ('n'), regular trigger to target ('r'), external command ('x'), or both regular trigger and external command ('b') - if unspecified, triggering is set by '-!
`-x<c|f|i|1>[p]`	On error, continue running ('c'), exit after current FOP ('f'), exit immediately at the point of error ('i') after the specified number of retries only if the retry fails, or exit immediately at the point of error after first retry (‘1’) whether or not the retry was successful - use the optional 'p' suffix to exit the program rather than just the affected I/O thread - if unspecified, the exit-on-error is set by '-!'
`- v<count>`	Set the retry 'count' (override global '-v')
`- p<hexstring>`	Set the trigger pattern to 'hexstring' (up to 16 hexadecimal characters) - if unspecified, the trigger pattern is set by '-!

Example: `pain --handler=read,write:tr,xip,pBAD,v3`

On read or write errors, regular trigger to target ('tr') using trigger pattern 'BADBADBADBADBADB' ('pBAD' - NOTE: 'BAD' is repeated to create a 64-bit patter integer), exit the program at the point of error ('xip') if still failed within 3 retries ('v3').

Example: `pain --handler=size,timeout,halt:lw,tr,xf`

Treat size error, timeout, and halt events as warnings ('lw'). The 'tr' and 'xf' are ignored. This option can be specified multiple times - all '--handler' specifications are combined to form the error event handling map.

Default:

Error handling is set using '-!'.

Chapter 5

Logging and Output

In this chapter:

Medusa Labs Test Tools (MLTT) provides detailed logs of performance and error conditions. The log files are written to the current directory from which the tools are executed. Logs are named according to the WS_NAME (workstation name) variable. The WS_NAME is either read from an environment of the same name, or the system's host name is used, if the environment variable is not found. When you use Catapult, it supplies the WS_NAME to each instance of MLTT it launches. The WS_NAME is a combination of the system host name and the target device name. For example, a host named *myhost* and a target of `\\physicaldrive2` would create a WS_NAME of `myhost_2`.

Four log files can be created during a test run:

- General status log (created by default)
- Performance summary log (created by default)
- Comma-delimited performance and error log (created by default)
- Error log (created when critical errors occur, on a per-thread basis. Each worker thread creates its own error log.)

Status Log

The general status log is named after the WS_NAME, with a .log extension, for example, *mysystem.log*. Information from a test run is always appended to the log file, so subsequent test runs will not overwrite the file. The status log records the following details about the test run:

- Start time
- Command line switches—with the settings for each switch, the settings you provided or the defaults
- Complete parameter and environment values
- Performance samples (the same samples that are displayed on the console screen during a test run)
- Error messages and counts

Figure 61 on page 188 shows a sample status log.

Performance Summary Log

The performance summary log is named after the WS_NAME with a .prf extension, for example *mysystem.prf*. This log file is overwritten with each performance sample. The log shows overall performance by listing the real-time readout. The performance summary log contains overall performance details for a test run, including the minimum, maximum, and average I/O operations (IOPS) and the minimum, maximum, and average MB/s. A count of total errors encountered is also listed. Catapult uses this file to verify test results. Figure 58 shows a performance summary log.

Figure 58: Performance Summary Log

```
pain 10 - 18 -b128k -o -l14 -!
Start Time=05/18/04 15:09:38
[Completion Info]
            Elapsed time=45.000   — Test time elapsed
            Samples=9             — Number of performance samples taken
            I/O Halts=0           — Number of times performance was equal to 0 MB/s
                                    in a sample
[I/O Operations]
            Total=6877            — Total IOPS in elapsed time
            Avg IO/Sec=152.82
            Max IO/Sec=153.40     — Average, Maximum, Minimum IOPS
            Min IO/Sec=152.60
[Megabytes]                       — Total MB/s in elapsed time
            Total=429.81
            Avg MB/Sec=9.55
            Max MB/Sec=9.59       — Average, Maximum, Minimum MB/s in elapsed time
            Min MB/Sec=9.54
[Errors]
            Total Errors=0
```

The performance summary log (.prf) is created with the same name every time a test case is run. So, the second and subsequent test cases would overwrite the contents of the performance summary log. In order to preserve the performance summary log for each test case, the batch file template, perfbaselinetest.bat, can be used. This batch file can be customized to individual needs: it essentially copies the performance summary log for each test case to a different name to avoid it being overwritten. In particular, the variables DEVICE and TARGET need to be customized.

The list of performance summary logs can be consolidated using the executable, prfgrab.exe, into a comma delimited file (.csv) that can be imported into Microsoft Excel for easy sorting and viewing. The executable, prfgrab.exe, is only available for Windows; there is no UNIX equivalent available. However, since .prf files are common across platforms, .prf files from a UNIX system can be brought into a Windows system where the prfgrab.exe executable can consolidate them into a .csv file.

Comma-delimited Performance Log

This log contains detailed performance samples and error counters that you can import into other programs for graphing or analysis. This log is named after the WS_NAME, with a .csv extension, for example, *mysystem.csv*. The .csv log file is appended with each test run. Figure 59 shows an example of some of the fields from a comma-delimited performance log after being imported into a spreadsheet application. It contains column headings and the performance samples.

Figure 59: Comma-delimited Performance Log

Elapsed Time	Total IOPS	Avg IOPS	Max IOPS	Min IOPS	Total MB/s	Avg MB/s	Max MB/s	Min MB/s
2	326	163	164	162	20.38	10.19	10.25	10.13
3	481	160.33	164	155	30.06	10.02	10.25	9.69
4	639	159.75	164	155	39.94	9.98	10.25	9.69
5	795	159	164	155	49.69	9.94	10.25	9.69
6	950	158.33	164	155	59.38	9.9	10.25	9.69
7	1107	158.14	164	155	69.19	9.88	10.25	9.69
8	1264	158	164	155	79	9.88	10.25	9.69
9	1423	158.11	164	155	88.94	9.88	10.25	9.69
10	1581	158.1	164	155	98.81	9.88	10.25	9.69
11	1739	158.09	164	155	108.69	9.88	10.25	9.69
12	1897	158.08	164	155	118.56	9.88	10.25	9.69
13	2056	158.15	164	155	128.5	9.88	10.25	9.69

Error Log

The error log is named in one of two manners:

- In a single threaded test it is named after the WS_NAME variable, with a .bad extension, for example *mysystem.bad*.
- In tests with multiple threads, it is named after the thread number where the error occurred, for example, *thread1.bad*.

This log contains pertinent details about the error encountered and is essential for debugging of data corruption issues. When a data corruption occurs, the entire bad data is listed along with the expected data and the offset counts where the differentiation in the comparison (the miscompare) occurred.

Sample Logs

This section contains sample logs generated by MLTT.

Sample Error Log

Figure 60 shows an annotated error log. To help you to evaluate the results, lines from one value to another show how the same information is presented in different formats.

Figure 60: Thread.bad Annotated Error Log


```
================================================================================
Wed Sep 10 16:24:45 2014: thread2.bad

"C:\Program Files\Medusa Labs\Test Tools\bin\pain.exe" -t4 -Y1 -! --target=\\.\PhysicalDrive1
================================================================================
09/10/14 16:24:45 Thread:2 Error:13 - Data corruption detected (LBA:0x2800 reqSize:65536 retSize:65536)
09/10/14 16:24:45 Thread:2 - writing trigger to target 1 '\\.\PhysicalDrive1' offset: 0x0F0000 LBA: 0x0780
09/10/14 16:24:45 Thread:2 - writing trigger to 'trigger.out'
09/10/14 16:24:45 Thread:2 Miscompare at 0x000000000000F800 bytes read in loop 67!

Miscompare: Offset: 0x000000000000F800 Wrote: 83FF83FE83FD83FC Read: 7C007C017C027C03
Dumping r/w buffers to: 'WIN-4C6HHBT72R1-0000000000500000.2w' and 'WIN-4C6HHBT72R1-0000000000500000.2r'...

Device: \\.\PhysicalDrive1
Offset:            Returned:                  Expected:
--------------------------------------------------------------------------------

0x000000000000F800  7C007C017C027C03 = |_|_|_|_  83FF83FE83FD83FC = ________
0x000000000000F808  9ACEFFFD43000001 = ___C___  6531000243000001 = e1__C___
0x000000000000F820  7C107C117C127C13 = |_|_|_|_  83EF83EE83ED83EC = ________

ERR: ------------------------------------------------------------------------
ERR: I/O Operation:        Read
ERR: Session ID:           0x6531 / 25905
ERR: Loop count:           67
ERR: Elapsed time:         00:00:00:05
ERR: File name:            \\.\PhysicalDrive1
ERR: Starting offset:      0x500000 / 5242880
ERR: Ending offset:        0x8FFFFF / 9437183
ERR: Base offset:          0x100000 / 1048576
ERR: * Note LBA values are valid only for physical device access.
ERR: Error LBA:            0x2800
ERR: Physical LBA range:   0x2800 to 0x47FF
ERR: Data pattern:         17 / 16-bit inc/dec pattern
ERR: Pattern cycle length: 1
ERR: Pattern direction:    Down/Odd
ERR: I/O direction:        Forward
ERR: Loops:                67
ERR: First expected signature
ERR: (or pattern in the first signature area if signature is disabled):
ERR:     65 31 00 02 43 00 00 01 00 00 28 00 FF F5 FF F4
ERR: Write buffer address: 0x01E70000 / 31916032
ERR: Read buffer address:  0x01E80000 / 31981568
ERR: I/O size (expected):  0x010000 / 65536
ERR: I/O size (actual):    0x010000 / 65536
ERR: Block number:         0x01 / 1
ERR: Block start:          0x0 / 0
ERR: Error start:          0xF800 / 63488
ERR: Error end:            0xF827 / 63527
ERR: Error length:         0x28 / 40

09/10/14 16:24:45 Thread:2 - retrying read on corruption (retry delay 0)

09/10/14 16:24:45 Thread:2 - no error on retry - LBA:0x2800
```

(*) This is a best guess at the LBA. Since I/O operations might be broken up into several small operations by the OS or drivers, the corrupt data might be in a different LBA. This LBA corresponds only to the start of the I/O block. Note that the LBA is only accurate for physical drives.

Sample Status Log

Figure 61, Figure 62,and Figure 63 show a sample status log.

Figure 61: Status Log (Page 1 of 3)

```
================================================================================
START: Wed Sep 10 16:24:40 2014
--------------------------------------------------------------------------------
Medusa Labs Test Tools 7.0.0.155614
Copyright (c) 1997-2007, 2008-2014, JDS Uniphase Corp. All rights reserved.
PAIN 3.4.0 (win-x64) Medusa Labs Synchronous I/O Test Tool
Support  : TechSupport-Medusa@jdsu.com
Built on : Wed Sep 10 19:32:23 UTC 2014
OS       : Windows Server 2008 R2 Datacenter Service Pack 1 (6.1.7601)
CPU      : Intel(R) Xeon(R) CPU E5506  @ 2.13GHz (Intel64 Family 6 Model 26 Stepping 5)
NCPUs    : 4 logical CPUs
MEM      : 3.990GB
VMWARE VM: NO
--------------------------------------------------------------------------------
Workstation name set to: WIN-4C6HHBT72R1
Initializing license management library...
Loaded: Medusa L.E.O. module for Medusa License Management client library
LM Server: 10.23.13.2
LM Port: 5033
License library initialized
Checking for existing license...
This system has a valid network checkout license
Licensed to Medusa Labs (valid until Mon Sep 15 14:59:33 2014)


Command line entered: "C:\Program Files\Medusa Labs\Test Tools\bin\pain.exe" -t4 -Y1 -! --target=\\.\PhysicalDrive1

Checking command line parameters and environment variables
Reading configuration from 'C:\Program Files\Medusa Labs\Test Tools\config\MedusaTools.cfg'

Checking targets

Command line parameters and environment processed:

    File size                = 4MB
 -A custom session ID        = OFF
 -b Buffer size              = 64KB
 -B I/O direction            = 0 (forwards only)
 -c Commit/flush             = OFF
 -C Compare mode             = 1 (full byte-by-byte)
 -d Test duration            = 0 (no limit)
 -D Pattern display          = OFF
 -e Blink length             = 0 bit(s)
 -E Hold time                = 0 cycle(s)
 -f Target                   = \\.\PhysicalDrive1
 -F Flip-flop mode           = OFF
 -g Burst interval           = 0 millisecond(s)
 -H Retry delay              = 0 second(s)
 -i Iterations               = 0 (no limit)
 -I Pattern invert           = OFF
 -j Pre-scramble             = 0 (off)
 -J Scrambler interval       = 1024
 -l Data pattern             = 17 (16-bit inc/dec pattern)
 -L Pattern cycles           = 1
 -m I/O mode                 = 1 (synchronous, sequential)
 -M Timeout monitor interval = 1 second(s)
 -n Disable data compare     = OFF
 -N Disable reverse pattern  = OFF
 -o Keep file handles open   = OFF
 -O Override base offset     = 3 (do not override base offset, error out on bad I/O extent)
 -P Phase shift              = 0
 -q Output level             = 0 (default logging and output level)
 -Q Maximum queue depth      = 1
 --scsi Direct SCSI CDB      = OFF
 -r Read-only                = OFF
 -R Read buffering           = 1 (cache allowed, no O/S buffering)
 -s Single sector read only  = OFF
 -S I/O thread/target startup delay = 0 (no thread delay); (no target delay)
 -t I/O threads per target   = 4
 -T Thread/CPU affinity      = Unbound (no affinity)
 -u Disable data signature   = OFF
 -U Timestamp the signature  = OFF
 -v Retry count on error     = 0 per I/O error, 1 per data comparison error
```

System Information

Command Line Entered by User

Switch Values Used for this session (user-specified and defaults)

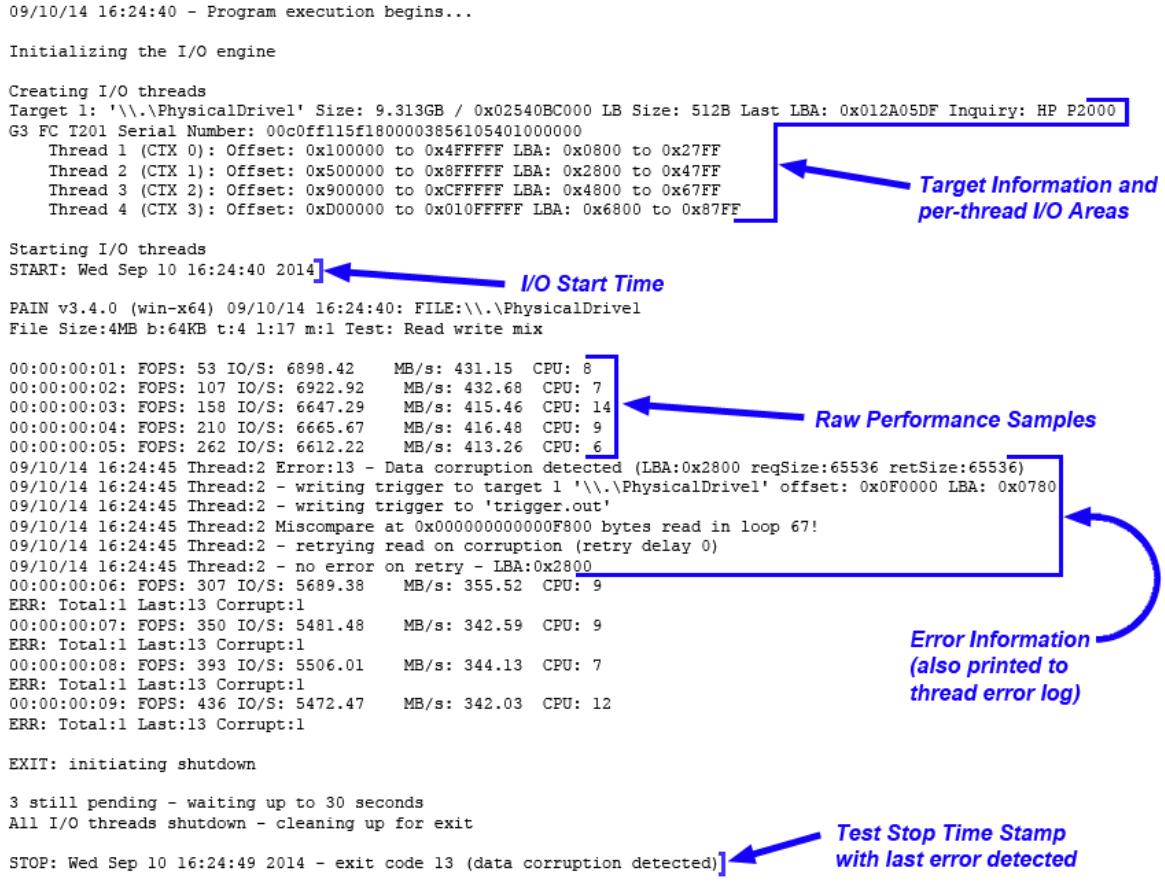
Figure 62: Status Log (Page 2 of 3)

```
 -V Reverify mode            = OFF
 -w Write only               = OFF
 -W Write buffering           = 1 (cache allowed, no O/S buffering)
 -x Base offset              = OFF
 -X Shared base offset       = OFF
 -y Data pattern seed value  = 0x0
 -Y Sample output interval   = 1 second(s)
 -! / -# Trigger mode        = 1 (0xCACACACA 0xCACACACA / 0xCACACACA 0xDEADBEEF)

 I/O profile:

 -%T Transaction mode  = OFF

I/O loop error event handlers:

Size (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Open (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Flush (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Close (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Read (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Write (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = at current offset
    Exit scope       = thread
    Retry count      = 0
Timeout (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = per target
    Trigger pattern  = CACACACADEADBEEF
    Trigger debugger = no
    Exit             = no
    Retry count      = 0
Halt (label=error)
    Trigger          = no trigger
    Trigger debugger = no
    Exit             = no
    Retry count      = 0
Corrupt (label=error)
    Trigger          = regular trigger output to target
    Trigger once     = no
    Trigger pattern  = CACACACACACACACA
    Trigger debugger = no
    Exit             = after current FOP
    Exit scope       = thread
    Retry count      = 1
```

I/O Profile Specification if -%r/w option is used (it is not used in this example)

Error Handling Disposition set by -! and --handler options
For additional information, refer to Appendix E Exit Codes

Figure 63: Status Log (Page 3 of 3)



Chapter 6

Data Pattern Reference

In this chapter:

Overview

The data pattern library built into the Medusa Labs Test Tools (MLTT) provides the basis for applying focused signal stress across a wide variety of architectures. There are several characteristics in our approach to data patterns which make them extremely effective.

Designed For Signal Aggravation

Some of the data patterns are industry standards which have been used for years in various I/O tools and traffic generators. Medusa Labs has also developed a number of patterns based on our test experiences. The data pattern library contains signal aggravating patterns suitable to most major I/O signal paths such as PCI, SCSI, Fibre Channel, and Ethernet. Rather than running random or empty data streams to the device under test, MLTT provides a means of stressing signal lines in a targeted, precise manner. This approach greatly increases the chance of identifying signal related defects in a timely manner.

Customized Patterns

You can customize the majority of the patterns to certain degrees for experimentation during a test effort. It is not uncommon to find issues with a slight modification of a pattern that a “stock” pattern failed to detect. You can set up scripted test runs to perform gradual variations on variables such as signal hold time on a bus. Additionally, you can customize some data patterns to suit testing on bus architectures of varying widths. This makes it easy to create relevant data patterns with the same MLTT as new hardware emerges.

Continuously Changing I/O Stream

A common deficiency in data integrity checking is that the data patterns utilized are static (that is, the same pattern is written repeatedly to the same target area.) This approach does not uncover data corruptions as a result of stale data being returned from the target. The Medusa Labs Test Tools overcome this problem by continually modifying the data stream, when possible, with each successive write. The methods that implement this do not result in any excessive overhead. Most of our data patterns are modified during run time in two ways.

1. Pattern Reversals – Most patterns in our library have a corresponding bit-for-bit reversal value. During a test, the data pattern is written out “forward” in one write pass and “reversed” in the next pass, continuously.

   Example (pattern number `-111`):

   Forward Pattern:

   ```
   0000FFFF
   0000FFFF
   0000FFFF
   0000FFFF
   ```

   Reverse Pattern:

   ```
   FFFF0000
   FFFF0000
   FFFF0000
   FFFF0000
   ```

2 I/O Signing – All of our data patterns have a unique signature added to the data at an interval of 512 bytes by default. This signature provides additional insurance that the data written is constantly changing. Additionally, these signatures serve as an important resource in debug efforts when a failure is encountered. The signatures are extremely useful in analyzing data captured by a protocol analyzer. See Appendix D, “I/O Signatures for more information about I/O signatures.

Customizing Data Patterns

Each data pattern in MLTT has a default form that is used when you use the `-l`*`pattern_number`* switch alone. This section covers the available switches that you can use to customize certain data patterns. Note that not every data pattern can be altered with the switches discussed here. Refer to the command line help for the currently supported data patterns for each of these options.

Using Pattern Modifiers

-L Number of Times to Repeat the Data Pattern Cycle

Usage:

`-Lcycle_length`

Use the -L*cycle_length* switch to modify the length or repetition of the selected data pattern's cycle. This switch is only supported with certain data patterns. Refer to the command-line help for current pattern support. The cycle length indicates the number of times to repeat each unit of a data pattern, before moving to the next unit. In general, the unit of data pattern refers to its length in bytes or bits.

Examples:

The following example is a data pattern comprised of random byte values. Here, the `-L4` modifier switch causes each byte to be replicated four times.

8-bit pattern: `pain -l35 -L4`

```
data: 0xCDCDCDCD
      0x59595959
      0x83838383
      0x7B7B7B7B
      …etc.
```

The following example is a data pattern comprised of a 16-bit incrementing value. Here, the `-L8` modifier switch causes each 16-bit value to be replicated eight times.

16-bit pattern: `pain -l17 -L8`

```
data: 0x00000000
      0x00000000
      0x00000000
      0x00000000
      0x00010001
      0x00010001
      0x00010001
      0x00010001
      …etc.
```

-I Invert Pattern Mode

The `-I` switch causes a bit inversion of the data pattern with each transition cycle. Refer to the MLTT command-line help for a listing of data patterns that support this option. This switch is useful for creating patterns for stressing signal lines on bus architectures.

Example:

The following example is a data pattern comprised of walking bytes, at 4-byte intervals. The `-I` switch is used to invert each walking value.

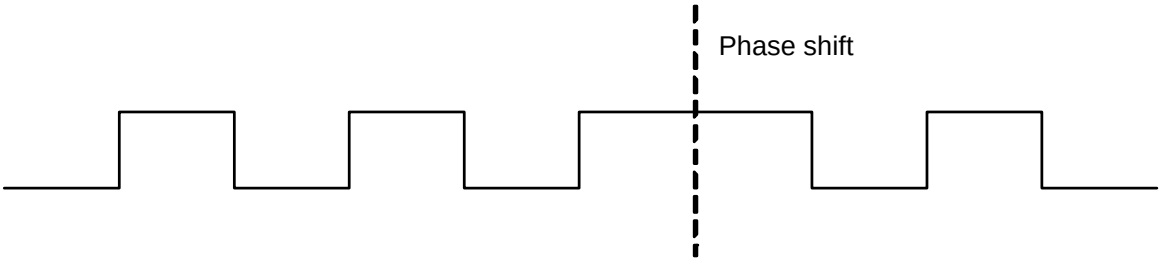
32-bit pattern: `pain -l10 -I`

```
data: 0x00000000
      0xFFFFFFFF
      0x01010101
      0xFEFEFEFE
      0x02020202
      0xFDFDFDFD
      …etc.
```

-P Modify Data Patterns with a Phase Shift

Use the `-P` switch to modify supported data patterns with a "phase shift." This switch works on most blinking data patterns. The effect is one of shifting the data pattern "out of phase" at the specified cycle length, such that the square wave created by the on/off bits reverses (Figure 64). Refer to the MLTT command-line help for a listing of data patterns that support this option.

Figure 64: Data Pattern Phase Shift



You can specify the frequency of this shift as the cycle length, or a default value is used if you enter `-P` only. A specified cycle length is multiplied with the pattern length (in bytes) to determine the frequency of the shift.

Example:

The following command line indicates a blinking 16-bit data pattern. Here, the `-P12` modifier switch causes a phase shift to occur every 12 bytes (l99 is treated as a one byte pattern length). Custom blinking pattern:

```
pain -l99 -L16 -P12

data: 0x0000FFFF
      0x0000FFFF
      0x0000FFFF
      0xFFFF0000
      0xFFFF0000
      0xFFFF0000
      0x0000FFFF
      0x0000FFFF
      0x0000FFFF
      …etc.
```

Custom Blink Pattern

Blinking bit patterns are a standard test data stream for bus and serial architectures. You can use a special data pattern, indicated by `-l99` to create a customized blink according to specified parameters. Specify the blink parameters with an optional walking bit argument on the data pattern switch itself and several additional switches.

You can specify the base data pattern switch as follows:

`-l99` = blink only, no walking bit

`-l99w` = adds bit walk to both “off” and “on” bit cycles)

`-l99o` = adds bit walk to “off” bit cycle only,

`-l99f` = adds bit walk to “on” bit cycle only.

The base data pattern is dependent on the -L*cycle_length* parameter, which is used to indicate the length of the blink in bits.

Examples:

8-bit blink: `pain -l99 -L8`

```
data: 0x00FF00FF
      0x00FF00FF
      …etc.
```

32-bit blink with full walk: `pain -l99w -L32`

```
data: 0x00000000
      0xFFFFFFFF
      0x80000000
      0x7FFFFFFF
      0x40000000
      0xBFFFFFFF
      …etc.
```

32-bit blink with “on” walk: `pain -l99o -L32`

```
data: 0x00000000
      0xFFFFFFFF
      0x80000000
      0xFFFFFFFF
      0x40000000
      0xFFFFFFFF
      …etc.
```

32-bit blink with “off” walk: `pain -l99f -L32`

```
data: 0x00000000
      0xFFFFFFFF
      0x00000000
      0x7FFFFFFF
      0x00000000
      0xBFFFFFFF
      …etc.
```

You can customize the blinking pattern further by using the -e*bit_length*, -E*hold_cycles*, and `-F` switches.

-e Custom Blink Pattern Modifier

Usage:

`-ebit_length`

This switch allows you to specify different bit lengths for the “on” bits. Use this switch in conjunction with the -L switch, which serves the special purpose of controlling the length of the “off” bits when used together with the `-e` switch.

Examples:

No bit walk: `pain -l99 -L28 -e4`

```
data: 0x0000000F
      0x0000000F
      …etc.
```

With bit walk: `pain -l99w -L8 -e24`

```
data: 0x00FFFFFF
      0x807FFFFF
      0x40BFFFFF
      …etc.
```

-E Custom Blink Pattern Modifier (for walking bit variations)

Usage:

`-E`*`hold_cycles`*

This switch allows you to specify a blink value to repeat for the number of hold cycles indicated, before walking a bit. This switch is valid with the data pattern switches `-l99w`, `-l99o`, and `-l99f`.

Example:

The following command line results in a 16 bit blinking data pattern that walks a bit every 2 cycles:

```
pain -l99w -L16 -E2

data: 0x0000FFFF
      0x0000FFFF
      0x80007FFF
      0x80007FFF
      …etc.
```

-F Custom Blink Pattern Modifier

Usage:

```
-F
```

This switch causes a “flip/flop” variation to occur within the blinking data pattern. By “flip/flop,” we mean that the pattern starts at an initial value, inverts (blinks) the value, returns to the initial value, then walks a bit and repeats the sequence. This switch is intended to be used with the data pattern switches `-l99w`, `-l99o`, and `-l99f`.

Example:

```
pain -l99w -L32 -F

data: 0x00000000
      0xFFFFFFFF
      0x00000000
      0x80000000
      0x7FFFFFFF
      0x80000000
      …etc.
```

Specified Data Patterns

In addition to the supplied base data patterns, it is possible to specify a particular pattern with command-line switches.

-y Create Data Patterns Based on Various Lengths

Usage:

`-ypattern_value`

Use the `-y` switch to specify a value to repeat in the write buffers and create the data pattern. You can use this switch with several data pattern numbers `(-l)` to create patterns based on various lengths. You can use this switch with the following data pattern numbers:

`-l1`

`-l2`

`-l4`

Example:

```
pain -l4 -y0x55AA55AA

data: 0x55AA55AA
      0x55AA55AA
      …etc.
```

-l0, -@*file_name* Read Data Pattern from a File

The `-l0` data pattern is a special pattern number that indicates that the data pattern is to be read in from an existing file. Use the `-@` switch to indicate the path and file that contains the data pattern. The file must be greater than or equal in size of the buffer size. The file is read up to the size indicated by the specified buffer size (`-b#`). If the file is larger than the buffer size, it will be continually read to fill the I/O buffers, so that large data patterns may be used effectively.

Example:

```
pain -b512k -l0 -@c:\data\pattern.dat
```

-@ Path and File of Data Pattern

Use the `-@` switch to indicate the path and file that contains the data pattern. The file must be greater than or equal in size of the buffer size.

Example:

```
pain -b512k -l0 -@c:\data\pattern.dat
```

Chapter 7

Catapult Test Tool Automation

In this chapter:

Catapult is a target discovery tool included with the test tool suite that acts as a shell for the I/O tools. You use Catapult to discover targets available to the host system and pass these targets to the other test tools for I/O testing. Catapult also has features that facilitate test scripting and automation.

Basic Usage

Catapult discovers available local or remote drives for file system, logical, or physical access. Available drives can be listed without starting an I/O test by using the following switches:

File System list: `catapult -f`

Logical drive list: `catapult -l`

Physical drive list: `catapult -p`

Remote drive list: `catapult -r` (use this switch with `-f`, `-l`, or `-p`)

Drive Listing Examples:

Figure 65 shows a listing of physical drives discovered on a Windows system. Figure 66 shows a listing of physical drives discovered on a Linux system. Figure 67 shows a file system drive discovery on a Windows system.

Figure 65: Catapult Physical Drive Discovery on Windows

Note: In order to run to physical devices, you must be logged in with administrator access.

```
C:\> catapult -p
Medusa Labs Test Tools 2.5.0
Catapult Version 1.3.5 Copyright (c)2004-2006, JDSU Inc.
All rights reserved.
Build date: Sep 12 2007 15:46:55

Searching for physical drives...

Available physical drives:

Indx Drive Name      SCSI ID  INQUIRY DATA                    Exclusions
-------------------------------------------------------------------------
  \\.\PhysicalDrive0  0:0:0:0  ST330630A               3.21    A  P
  \\.\PhysicalDrive1  2:0:0:0  MAXTOR  ATLASU320_18_SCAB430    A  S
  \\.\PhysicalDrive2  2:0:1:0  MAXTOR ATLASU320_18_SCAB430
  \\.\PhysicalDrive3  4:0:0:0  SEAGATE ST39103FC       0004
  \\.\PhysicalDrive4  4:0:1:0  SEAGATE ST39103FC       0004
  \\.\PhysicalDrive5  4:0:2:0  SEAGATE ST39103FC       0004
  \\.\PhysicalDrive6  4:0:3:0  SEAGATE ST39103FC       0004
  \\.\PhysicalDrive7  4:0:4:0  SEAGATE ST39103FC       0004
  \\.\PhysicalDrive8  4:0:5:0  SEAGATE ST39103FC       0004
```

Figure 66: Physical Drive Discovery on Linux

Note: In order to run to physical devices, you must be logged in with root access.

```
catapult -p

FOUND: Bootdev=/dev/sda

Medusa Labs Test Tools 2.5.0
Catapult Version 1.3.5 Copyright (c)2004-2006, JDSU Inc.
All rights reserved.
Build date: Sep 12 2007 15:46:55

Indx Device Name SCSI ID Inquiry Data                  Exclusions
-------------------------------------------------------------------
00   /dev/hda Not Scsi VMware Virtual IDE Hard Drive
01   /dev/hdb Not Scsi VMware Virtual IDE Hard Drive
02   /dev/sda 0:0:0:0 VMware, VMware Virtual S 1.0         PS
03   /dev/sdb 0:0:1:0 VMware, VMware Virtual S 1.0
04   /dev/sdc 0:0:2:0 VMware, VMware Virtual S 1.0
05   /dev/sdd 0:0:3:0 VMware, VMware Virtual S 1.0
06   /dev/sde 0:0:4:0 VMware, VMware Virtual S 1.0
07   /dev/sdf 0:0:5:0 VMware, VMware Virtual S 1.0
08   /dev/sdg 0:0:6:0 VMware, VMware Virtual S 1.0
```

Figure 67: File System Drive Discovery on Windows

```
C:\> catapult -f
Medusa Labs Test Tools 2.5.0
Catapult Version 1.3.5 Copyright (c)2004-2006, JDSU Inc.
All rights reserved.
Build date: Sep 12 2007 15:46:55

Available file systems:

Indx Drive FileSys Size(MB) Label/Mount         Exclusions
-----------------------------------------------------------
     C:\    NTFS    32171   (unlabeled disk)       A  PS
     D:\    NTFS    47829   (unlabeled disk)
```

The output for file system and logical drives is the same on Windows platforms, but I/O test access is different. Logical access utilizes destructive “raw” partition access. File system access is restricted to data files on a file system partition.

To run I/O to discovered targets, the desired I/O tool and its switches are passed to Catapult on the command line, immediately following switches used for Catapult.

Note: When you use Catapult to start I/O tests, do NOT specify a target (-f switch) as a Test Tool argument. Catapult will supply this switch to each Test Tool instance for you.

Warning: It is important to understand that by default, ALL drives listed WILL be included in I/O tests unless they are excluded due to a reason stated in the listing output. You must use the drive include or exclude switches discussed in the following sections if you want to test only certain drives.

Warning: Logical and physical drive access is destructive! Any existing data on the drives WILL be destroyed by I/O tests!

Catapult Command Example:

The following command launches an instance of Pain on each discovered physical drive.

```
catapult -p pain -t10 -b128k -o
```

By default, Catapult runs tests on all eligible drives of the specified type (physical, logical, or file system). Target access can be limited with the inclusion/exclusion switches described in “Catapult Switches” on page 207.

Note: Catapult will launch an instance of the specified test tool to all eligible drives that are detected. To prevent overwriting critical data, the test tool performs several checks to verify drive eligibility. I/O testing is skipped on drives that do not pass the checks. Catapult skips drives with active (bootable) partitions, drives with no volume label, and the drive where the operating system is installed. On Windows platforms, drives A:- C: are excluded and logical drive access requires that a drive letter be assigned.

In order to keep log files for each target separate, Catapult creates a new directory for each tool instance in the current working directory. The new directory name is derived from the system or host name and the target device (for example, winhost_1 for `\\.\physicaldrive1`, winhost_F for `\\.\F`, etc.) Each session of the I/O tools is launched from its respective directory so that each session’s log files are stored in a uniquely identified working directory.

Catapult Switches

This section contains a complete listing of the Catapult command line switches in alphabetical order. You can combine multiple switches, but you must enter all Catapult switches before you enter the I/O tool name and its switches.

- -a Auto-mode
- -b Log retrieval
- -c Clean Directories
- -d Delay Test Start
- -e Increment Data Pattern
- -f File system access
- -g Change directory prefix
- -h online help
- -i Include drive (also -i.str_Include drive based on inquiry data)
- -j Limit inquiry ioctls
- -k Kill tool processes
- -l Logical drive access
- -m Minimize tool windows
- -n Enable prompts
- -now Run all tests with no windows
- -o Override device exclusions
- --off Offline disk
- --on Online disk
- -p Physical drive access
- -q Removes excluded drives
- -r remote access
- --restart-service Restarts the Medusa agent
- -s Set tool starting offset
- -t Multi-target mode
- -v Verify mode
- -w Watch mode
- -x Exclude drive (also -x.str_Exclude drive based on inquire data)
- -y Specify grace period
- -z Debug mode

-a Auto-mode

Usage:

`-a`*`seconds`*

Description:

Use the `-a` switch to run tests for the duration specified in seconds. Use this switch primarily in scripted test runs. The test tool runs for the specified number of seconds, after which Catapult terminates all instances of the tool. Catapult will remain running during the I/O test. If Catapult is stopped (for example, by using **Ctrl+C**), Medusa Labs Test Tools (MLTT) will not be terminated when the time duration expires.

Example:

The following example runs an instance of Pain to each detected physical drive for a period of 300 seconds.

```
catapult -a300 -p pain -t10 -b128k -o
```

Default:

There is no default value for seconds to run; you must supply a value.

-b Log retrieval

Usage:

`-b -r`*`server`*

Description:

Use the `-b` switch to retrieve logs from the servers that you specify with the `-r` switch. This will retrieve only the specified and/or used subdirectories from the default testing location on the remote system. If the local system already has some of the directories or logs, you will be prompted to overwrite existing data. To force the overwrite without a prompt, use `-b!`.

Example:

The following example finds the logs on the remote server.

```
catapult -b -rserver
```

Default:

There is no default log retrieval. You must specify servers with the `-r` switch, along with the `-b` switch.

-c Clean Directories

Usage:

```
-c
```

Description:

Use the -c switch to clear out the working directories used in tool sessions created by Catapult. You must specify a drive type switch (-f, -l, or -p) along with the -c switch. Catapult removes ALL files in the working directories before launching the specified test tool. Use this switch a single time prior to running a scripted test.

Caution: Do not include this switch during a scripted run. Removing log files makes it impossible to determine test success or failure. Make sure that you have backed up any log files that you want to keep before using this switch!

Example:

The following example deletes all files in the working directories before running an instance of Pain to each detected physical drive.

```
catapult -c -p pain -t10 -b128k -o
```

You can also clear log files without running a test tool as shown in the following example:

```
catapult -c -p
```

Default:

There are no additional arguments. The contents of working directories for the specified drive type (file system, logical, or physical) are cleared.

-d Delay Test Start

Usage:

`-d`*`seconds`*

Description:

Use the `-d` switch to create a delay between tool sessions created by Catapult. By default, each tool instance is started simultaneously. For example, 10 detected drives would result in 10 instances of Pain being started at the same time. Depending on variables such as drive count, thread count, and buffer size, you might want to use this switch to create a ramp-up period to lessen the initial "shock" to the host system or target.

Example:

The following example runs an instance of Pain with a 10 second delay between each launch to detected file system drives.

```
catapult -d10 -f pain -t10 -b128k -o
```

Default:

There is no default value for seconds; you must supply a value.

-e Increment Data Pattern

Usage:

`-e`

Description:

Use the `-e` switch as a scripting aid to increment data pattern numbers. This switch is supported on Windows platforms only. Use this switch within a batch file that steps through various data pattern variations. Catapult reads the environment variable PATTERN and creates a batch file (incdat.bat) that can be called from a script to increment the value of the variable by 1.

Example (Windows batch file):

The following batch file example runs an instance of Pain for 60 seconds to detected physical drives with varied parameters on each run. The data pattern is incremented with each loop through the batch file.

```
set PATTERN=10
:TOP
catapult -a60 -p pain -t10 -b16k -o -l%PATTERN%
catapult -a60 -p pain -t10 -b32k -o -l%PATTERN%
catapult -a60 -p pain -t10 -b64k -o -l%PATTERN%
catapult -a60 -p pain -t10 -b128k -o -l%PATTERN%
catapult -e
REM incdat.bat is created by catapult -e
Call incdat.bat
if '%PATTERN%' == '20' set PATTERN=10
goto TOP
```

Default:

This switch increments the environment variable PATTERN by 1.

-f File system access

Usage:

```
-f
```

Description:

Use the `-f` switch to have Catapult scan for targets with file systems. When you use this switch without an argument, Catapult simply lists detected targets. When you specify a test tool, Catapult launches the tool on the detected targets. The tools will create data files for I/O traffic.

When Catapult detects targets, some devices may be excluded for the following reasons:

A : Active (possibly a boot device)
G : No signature or Bad/unlabeled VTOC
I : Inquiry failed
M: No media or insufficient memory
P : Partitions found on device (Windows physical disks only)
S : System device (where the OS is installed)
U : Excluded by use of inclusion/exclusion options
V : Device belongs to a volume group
W: Device is flagged as swap space or in a volume group that is flagged as swap space
Z : Device or path to device is inaccessible

Example:

When you execute the following switch, it lists all detected file system drives.

```
catapult -f
```

The following example runs an instance of Pain to all detected file system drives.

```
catapult -f pain
```

Default:

There is no default access method. You must specify a file system, logical, or physical access switch.

-g Change directory prefix

Usage:

```
-gdirectory_prefix
```

Description:

Use the -g to change the prefix of the directories used by the tools. Catapult uses the system host name for the directory prefix by default.

It will either prepend the directory with the prefix or it will create a folder.

Example:

`catapult -p -gtest1 pain` will prepend test1 to the folder

`catapult -p -gc:\test1` will create a folder called test1 to put the files in.

Default:

Catapult will use the system host name for a working directory prefix.

-h online help

Usage:

```
-h [-option]
```

Description:

Use the `-h` switch to display online help.

You can also use the -h switch to display online help for a specific option.

Note: -? can be used as an alternative to using the -h command to display the online help.

Example:

```
catapult -h
```

This example would show online help for Catapult.

```
catapult -h -t
```

This example would show online help for Catapult's -t option.

-i Include drive

Usage:

`-i`*`include_list`*

Description:

Use the `-i` switch to explicitly include drives for I/O testing. Drives not listed with this switch are excluded from testing. List drives by number for physical access and by letter for file system access on Windows platforms. On UNIX platforms, a number is listed next to each detected drive. List individual drives in comma-delimited format. You can indicate ranges by using a dash (-).

Use the '`.`' function to specify drives to include by inquiry, excluding all other targets.

`-i.<str>` would specify drives to include by inquiry, excludes all other targets.

Examples:

The following example runs an instance of Pain to the specified physical drives.

```
catapult -i1,2,3,5-10 -p pain
```

The following example runs an instance of Pain to the specified file system drives.

```
catapult -if,g,h-k,z -f pain
```

An example of inclusion/exclusion based on inquiry would be: If the user wishes to include/ exclude devices based on specific information returned during and inquiry (i.e. manufacturer or model number), this information is placed after the dot; as in

```
catapult -p -i.Seagate
```

would include any drives that returned the string "Seagate" in their inquiry response.

Default:

All drives are included by default. You must include or exclude the switches to customize the list of drives accessed in testing.

-j Limit inquiry ioctls

Description:

Use the -j option to specify input/output control timeout limit in seconds

Example:

This example sets the IOCTL timeout value to 2 seconds.

```
catapult -j2 -p pain -t10 -b128k -o
```

Default:

By default, the timeout value is 1 second.

-k Kill tool processes

Usage:

```
-k [-r hosts tool]
```

Description:

Use the -k switch to have Catapult send a kill signal to ALL running MLTT processes. This switch will cause ALL running tests to stop immediately and exit ALL MLTT processes.

You can also use the -k switch to kill ALL MLTT processes running on specific hosts.

Example:

```
catapult -k
```

```
catapult -k -rserver_name
```

Default:

This switch when used alone will kill ALL running MLTT processes.

-l Logical drive access

Usage:

```
-l
```

Description:

Use the `-l` switch to cause Catapult to scan for logical (partitioned) drives.

Warning: Logical drive access is destructive! Any existing data on the partition WILL be destroyed by I/O tests!

This switch is primarily for tests against targets such as an operating system RAID set. A common usage would be to utilize a striped set of drives (RAID 0), accessed as a raw partition, for HBA performance testing. When you use this switch without an argument, Catapult simply lists detected targets. When you specify a test tool, Catapult launches the tool on the detected targets.

Note: In order to run to logical devices, you must be logged in with administrator (Windows) or root (Unix) access.

When Catapult detects targets, some devices may be excluded for the following reasons:

A : Active (possibly a boot device)
G : No signature (Windows physical devices only)
I : Inquiry failed
M: No media or insufficient memory
P : Partitions found on device (Windows physical disks only)
S : System device (where the OS is installed)
U : Excluded by use of inclusion/exclusion options
V : Device belongs to a volume group
W: Device is flagged as swap space or in a volume group that is flagged as swap space
Z : Device or path to device is inaccessible

Example:

The following example lists all detected logical drives.

```
catapult -l
```

The following example runs an instance of Pain to all detected logical drives.

```
catapult -l pain
```

Default:

There is no default access method; you must specify a file system, logical, or physical access switch.

-m Minimize tool windows

Usage:

```
-m
```

Description:

Use the -m switch to have Catapult create instances of launched MLTT in minimized windows. This is useful when working with other applications, such as a performance monitoring utility.

Example:

The following example runs an instance of Pain to all detected physical drives, with each Pain window minimized.

```
catapult -m -p pain
```

Default:

By default, tool instances created by Catapult appear in open windows.

-n Enable prompts

Usage:

```
-n
```

Description:

Use the -n switch to indicate that Catapult should prompt the user if error logs (*.bad files) from a previous test are discovered in the tool working directories. Catapult will run a check for error logs before starting the I/O tool. This switch can be useful in scripted tests to prevent the script from continuing after the I/O tool encounters critical errors.

Example:

The following example runs an instance of Pain to all detected physical drives with this prompting if error logs are discovered.

```
catapult -n -p pain
```

If error logs are found, you will be prompted for the action to take:

```
Found existing error log: C:\test\VMW2KAS_1\VMW2KAS_1.bad
Delete log? ((Y)es/(N)o/Yes(A)ll/N(o)All):
```

Default:

By default, Catapult does not check for error logs.

-now Run all tests with no windows

Usage:

```
-now
```

Description:

Use the -now switch to run all tests with no windows (i.e. in the current console). The output can be piped to scripts, but if multiple systems are specified with '-r', the output will be jumbled. This option turns on '-t' multitarget option.

Example:

The following example will run the tests without displaying them in the console window.

```
catapult -now
```

Default:

No default is specified. Catapult will attempt to create new console or terminal windows for the test processes.

-o Override device exclusions

Usage:

```
-o
```

Use the -o switch to allow testing on volumes that have been automatically excluded. Using this option without include/exclude options will run to all attached devices except for devices marked as system (S). Specific overrides can be done by letter, i.e. to run to active devices but still keep other exclusions, use -oa. Please note that the system exclusion cannot be overridden.

Warning: This switch should be used with extreme caution! You can accidentally lose data. We highly recommend that you allow Catapult to perform drive signature checking. Use Windows Disk Management to assign drive signatures to unsigned drives.

Example:

The following example runs an instance of Pain to all detected physical drives with the drive signature check disabled.

```
catapult -o -p pain
```

Default:

By default, Catapult checks physical drives for a valid drive signature on Windows platforms.

--off Offline disk

Description:

In Windows Server 2008 or later, use the `--off` option to set drives to the 'offline' state. The tools cannot run on drives that are offline. This allows selected drives to be excluded from running in a test.

Example:

The following example sets all physical drives to **Offline**.

```
catapult -p -o --off
```

The following example sets the 2nd and 3rd physical drives to **Offline**.

```
catapult -p -i2,3 -o --off
```

--on Online disk

Description:

In Windows Server 2008 or later, use the `--on` option to set drives to the 'online' state when the drives are in 'offline' state. The Tools cannot run on drives that are offline.

Example:

The following example sets all physical drives to **Online**.

```
catapult -p -o --on
```

The following example sets the 2nd and 3rd physical drives to Online.

```
catapult -p -i2,3 -o --on
```

-p Physical drive access

Usage:

```
-p
```

Description:

Use the -p switch to have Catapult scan for physical drives.

Warning: Physical drive access is destructive! Any existing data on the drive WILL be destroyed by I/O tests.

This is the most common test access method used in hardware tests, as it bypasses as many layers of the operating system as possible. On Linux platforms, block devices will be automatically bound to “raw” devices by Catapult. When you use this switch without an argument, Catapult simply lists detected targets. When a test tool is specified, Catapult launches the tool on the detected targets.

Note: In order to run to physical devices, you must be logged in with administrator (Windows) or root (Unix) access.

When Catapult detects targets, some devices may be excluded for the following reasons:

A : Active (possibly a boot device)
G : No signature (Windows physical devices only)
I : Inquiry failed
M: No media or insufficient memory
P : Partitions found on device (Windows physical disks only)
S : System device (where the OS is installed)
U : Excluded by use of inclusion/exclusion options
V : Device belongs to a volume group
W: Device is flagged as swap space or in a volume group that is flagged as swap space
Z : Device or path to device is inaccessible

Example:

The following example lists all detected physical drives.

```
catapult -p
```

The following example runs an instance of Pain to all detected physical drives.

```
catapult -p pain
```

Default:

There is no default access method; you must specify a file system, logical, or physical access switch.

-q Removes excluded drives

Description:

Use the -q option to hide the excluded drives from being listed.

Example:

This example will list all physical drives but hide those that have device exclusion

```
Catapult -p -q
```

-r remote access

Usage:

```
-r
```

Description:

Use the -r switch to have Catapult scan for remote systems running MLTT.

This switch allows Catapult to start and stop tests on remote systems. MLTT must be previously installed on the remote systems. When you use this switch without an argument, Catapult simply lists all detected remote systems on the current subnet. When used with a basic switch (-f, -l, -p) Catapult lists the remote targets for those categories. In order to run a test, you must specify one or more system names. You can specify system names or IP addresses as a comma delimited list. You can also use the asterisk (*) as a wildcard character to specify systems names. When a test tool is specified, Catapult launches the tool on the detected remote targets. The -i and -x switches may used to include or exclude specific remote targets on specified systems.

Running `catapult -r`, also creates the file `hosts.dat`. This file will contain a list of IP addresses for the discovered host systems running MLTT. This file can be used in later test sessions to start remote tests by running `catapult -r[path]hosts.dat`. You can also use wildcards to customize the file creation. For example:

`catapult -rde*` would create a `hosts.dat` file with host names starting with '`de`.'

`catapult -r10.10.0.*` would create a `hosts.dat` file with hosts whose IP addresses start with `10.10.0`.

A `hosts.dat` file may be created manually by creating a text file with one host name or IP address per line.

Results from the -r Catapult scan are stored in log files saved in the MLTT installation directory on the remote systems in the sub-directory named “catapult_tests.”

- For Windows systems the path is:
  `c:\program files\JDSU\medusa labs\test tools\catapult_tests\`
- For Unix systems the path is:
  `/opt/medusa_labs/test_tools/catapult_tests/`

Use the -b switch to copy the remote files to a local system.

Note: In order to run to physical devices remotely, you must be logged in to the remote system with administrator (Windows) or root (Unix) access.

When using wildcards with some UNIX shells, you have to enclose the wild-carded system names in quotes, for example, `catapult -r 'sys*'`

Example:

The following example lists all detected remote systems.

```
catapult -r
```

The following example lists all detected remote physical drives on the specified server.

```
catapult -p -rserver_name
```

The following example runs an instance of Pain to all detected remote physical drives.

```
catapult -p -rserver_name pain
```

The following example runs an instance of Pain to the specified drives on the specified servers.

```
catapult -p -rserver1,server2 -iserver1:1,2 -iserver2:1 pain
```

The following example shows the use of the wildcard character to run an instance of Pain to all drives that begin with the name “medusa” or the IP address of 10.22.0 on the specified servers.

```
catapult -p -rmedusa*,10.22.0* pain
```

Default:

There is no default access method; you must specify a remote file system, logical, or physical access switch.

--restart-service Restarts the Medusa agent

Description:

Use the `--restart-service` switch to restart the Medusa agent. If you use this to restart the agent, it will interrupt process monitoring functions, so do not use this switch while you are running tests. This should be used as an option during a troubleshooting process if a particular system is showing problems, such as communicating with local or remote MLTT installations or if the GUI does not start up properly, etc.

Caution: If the `--restart-service` switch is used, tests that are currently running will be interrupted.

Example:

The following example restarts the Medusa agent.

```
catapult --restart-service
```

-s Set tool starting offset

Usage:

`-soffset_amount`

Description:

Use the `-s` switch to have Catapult pass a starting offset parameter (`-x`) to each instance of MLTT. The starting offset value for each launched instance of MLTT is incremented by the offset value (in megabytes (MB)) provided with this switch. You can use this switch to debug test configurations involving multiple initiator systems on a common set of targets. The offset amount must be a value that is greater than the number of worker threads run by the tools times the file size. For example, with 10 threads and a 10 MB file size, the offset amount needs to be at least 100 MB. See "-x Multi-Share Mode 1 - Multiple Sessions Offset" on page 167, for more information about the MLTT `-x` switch.

Note: -s<offset> ignores any user-supplied units and only uses MB as the units.

Example:

The following example runs an instance of Pain to all detected physical drives. Each instance of Pain has the `-x` switch appended to the command line. The offset value of the -x switch is incremented by 10 with each successive instance of Pain. With each Pain instance passed, a -x switch is incremented by 10.

```
catapult -s10 -p pain -t4 1
```

The launched instances of Pain have the following command lines:

```
pain -t4 1 -x0

pain -t4 1 -x10

pain -t4 1 -x20

…etc.
```

Default:

There is no default value for the offset amount; you must specify a value.

-t Multi-target mode

Usage:

```
-t (Use with -f, -l, or -p to set target type.)
```

Description:

The -t switch is used to indicate that Pain or Maim will run to multiple targets in a single process. By default, the I/O tools will run a single process per target. A new working directory with the identifier “MultiTarget” in the directory name will be created for the I/O tests to run in. This switch will change the log output format of the tools. The main tool performance output will display aggregate throughput for all targets in use. The general log file (*.log) will contain both aggregate numbers and individual target numbers for each performance sample. This switch essentially creates a list of targets in a file called “targets.dat” and passes this file as the argument for the I/O tool’s -f switch. If a test tool is not specified on the Catapult command line, the targets.dat file will be created without starting any I/O tests.

Example:

The following example runs a single instance of Pain with all detected physical drives as targets.

```
catapult -p -t pain -t4 -b4k -o 1
```

Default:

Use the -t switch to override Catapult’s default behavior of running a tool process per target. The target type (-f, -l, or -p) must be specified as well.

-v Verify mode

Usage:

```
-v
```

The -v switch can be used to perform a quick verification of a test run. This switch takes a number of other arguments that you can use as criteria for a pass/fail check of a test run. Catapult uses the performance summary log (*.prf) file created by MLTT for this check. Because this file is recreated by each test run, this check must be performed immediately after each test run that you want to check. If a verification check fails, Catapult will display a prompt and wait for user intervention. However, if you would like to run a verification on existing log files (post processing method), the -vp argument will scan any existing *.csv files instead of the *.prf files. The *.csv files are continuously appended to and can encompass multiple test runs.

The arguments you use with the -v switch are as follows:

`l`	Disables logging to `vlog.log` (enabled by default). The vlog.log is created in the current working directory and contains the same output that you see on the screen during a verification.
a*`Megabytes_sec`*	Minimum average MB/s allowed.
b*`IO_sec`*	Minimum average IO/s allowed.
i*`Halt_num`*	Maximum number of I/O halts to allow.
m*`Megabytes_sec`*	Minimum MB/s allowed.
n*`Megabytes_sec`*	Minimum IO/s allowed.
`e`	Check for any errors in test run.
`p`	Use post-processing method. This will search all subdirectories of the current directory and process .csv files that are of the form MLTT uses. Use this argument in combination with other -v arguments.
p*file_name*	Use this argument to process a specific .csv file by specifying its name.
`q`	Quiet mode. Only summary information is printed.
v	Verbose mode. More detailed information is printed.

Specify each argument individually after the -v switch as shown in the following example.

Example:

The following example will verify the performance logs in the working directories used for physical drive tests.

```
catapult -p -vm5 -va10.5 -vi0 -ve
```

The verification will report a failure and prompt the user if:

- The minimum MB/s for the test run was below 5 MB/s.
- The average MB/s for the test run was below 10.5MB/s
- One or more I/O halts occurred.
- Any error conditions occurred.

Default:

There are no default verification checks. The checks must be specified with additional arguments.

-w Watch mode

Usage:

`-wwait_seconds`

Description:

Use the `-w` switch primarily in conjunction with the auto mode (`-a`) switch. When you include this switch, Catapult monitors the test tool working directories for the creation of any error logs. If it detects error logs, Catapult terminates all instances of the running test tool after the specified wait seconds have elapsed. We recommend that you specify a reasonable number of wait seconds to allow for any tool instances that might be dumping error details to a file to complete. Under some circumstances, you might want a more immediate test termination. This switch is useful for cases where you are using a protocol analyzer to capture an error condition and want to avoid overrunning a capture buffer.

Example:

The following example runs an instance of Pain to all detected physical drives for 10 minutes. Catapult watches the working directories for the appearance of any error logs. If error logs are detected, Catapult waits for 30 seconds to allow time for error logs to be completed; then all instances of Pain are terminated.

```
catapult -w30 -a600 -p pain
```

Default:

The default value for the seconds to wait is 1. We recommend a higher value under normal test circumstances.

-x Exclude drive

Usage:

`-xexclude_list`

Description:

Use the -x switch to explicitly exclude drives from I/O testing. All drives that you list with this switch are excluded from testing. All other detected drives are included in testing. List the drives by number for physical access and letter for file system access on Windows platforms. On UNIX platforms, a number is listed next to each detected drive. List individual drives in a comma-delimited format. You can indicate ranges by using a dash (-).

Use the '.' function to specify drives to exclude by inquiry, including all other targets.

`-x.<str>` would specify drives to exclude by inquiry, includes all other targets.

Examples:

The following example runs an instance of Pain to detected physical drives and excludes the specified drives following the -x switch.

```
catapult -x1,2,3,5-10 -p pain
```

The following example runs an instance of Pain to detected file system drives and excludes the specified drives following the -x switch.

```
catapult -xf,g,h-k,z -f pain
```

An example of inclusion/exclusion based on inquiry would be: If the user wishes to include/ exclude devices based on specific information returned during and inquiry (i.e. manufacturer or model number), this information is placed after the dot; as in

```
catapult -p -x.Seagate
```

This example will exclude any drives that returned the string "Seagate" in their inquiry response.

Default:

All drives are included by default. You must use the include or exclude switches to customize the list of drives accessed in testing.

-y Specify grace period

Description:

Use the -y option to modify the grace period used in a timed test. Grace period is the time that time catapult gives the test process (pain or maim) to exit on its own after the test is done. After that time period, if the test process still has not exited, catapult will try to forcibly terminate the test process.

Example:

In the following example, the "`-a300`" command tells catapult to launch the pain test process to run for 300 seconds. Pain should run for 300 seconds and exit gracefully. After 300 seconds, catapult waits for 10 seconds ("`-y10`") for the pain process to exit. If pain process still has not exited after the specified 10 seconds, catapult will try to terminate the process.

```
catapult -a300 -y10 -p pain -t10 -b128k -o
```

Default:

By default, the grace period is 20 seconds.

-z Debug mode

Usage:

```
-z
```

Description:

Use the `-z` switch to instruct Catapult to create a debug log file (`catapult.dbg`) in the current working directory. You can use this log file to troubleshoot run time issues.

Example:

The following example runs an instance of Pain to all detected physical drives and creates a debug file.

```
catapult -z -p pain
```

Default:

Debug mode is disabled by default.

Scripting

The command line interface of MLTT is very conducive to scripted test runs. Catapult facilitates the creation of scripts that provide a broad range of test coverage in just a few lines. The following examples show Catapult scripts for Windows batch files.

Note: It is important to take the time following a scripted test run to examine the log files generated for any errors or anomalies.

Example 1 (Windows batch file)

The following example is a Windows batch file that will launch I/O tests using Pain on all detected physical drives. The duration of each test variation is set to 600 seconds. Different data patterns are run with various block sizes for the duration period. The data pattern number is incremented up one number by Catapult after a complete pass through all block sizes. When data pattern number 31 is reached, the test cycle is ended.

Refer to “-e Increment Data Pattern” on page 211 for more information about the PATTERN environment variable.

```
REM Data pattern / block size variations
set PATTERN=1
:START
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b4k
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b8k
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b16k
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b32k
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b64k
catapult -p -a600 pain -o -l%PATTERN% -t8 -! 1 -b128k
catapult -e
REM incdat.bat is created by catapult -e
call incdat.bat
if '%PATTERN%'=='31' goto STOP
goto START
:STOP
```

Example 2 (Windows batch file)

The following example is a Windows batch file that will launch I/O tests using Pain on all detected physical drives. The duration of each test variation is set to 600 seconds. Variations of the Custom Blink Pattern (-l99) are used to induce signal stress on a 64 bit PCI bus. See “Custom Blink Pattern” on page 198.

```
REM 64 bit blinking bus variations
:START
catapult -p -a600 pain -o -l99 -L64 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99w -L64 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99o -L64 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99f -L64 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e4 -L60 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e8 -L56 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e16 -L48 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e48 -L16 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e56 -L8 -t8 -! 1 -b128k
catapult -p -a600 pain -o -l99 -e60 -L4 -t8 -! 1 -b128k
:STOP
```

Appendix A
Data Pattern Numbers

This appendix lists the data pattern numbers and their descriptions as they are listed in the Pain and Maim help.

No.	Description
0	Read pattern from data file (specify file with -@filename)
1	8-bit value - Fixed pattern (specify value with -y0x##). If -y is not specified or set to 0, a default value is chosen
2	16-bit value - Fixed pattern (specify value with -y0x####). If -y is not specified or set to 0, a default value is chosen
3	32-bit value - Fixed pattern - 1st FOP 0xFFFFFFFF, 2nd:0x00000000
4	32-bit value - Fixed pattern (specify value with -y0x########). If -y is not specified or set to 0, a default value is chosen
5	32-bit value - Noise pattern #1 - Checkerboard
6	32-bit value - Noise pattern #2 - Blinking bus
7	32-bit value - Noise pattern #3 - Walking/XOR bits
8	8-bit value - Walking pattern 0x00 - 0xFF - 0x00
9	8-bit value - Walking double 0x0000, 0x0101, 0x0202...etc
10	8-bit value - Walking quad 0x00000000, 0x01010101...etc
11	16-bit value - Noise pattern #4 - SCSI Blinking bus
12	16-bit value - Noise pattern #5 - SCSI Alternating parity bus
13	64-bit value - Noise pattern #6 - Checkerboard
14	64-bit value - Noise pattern #7 - Blinking bus
15	64-bit value - Noise pattern #8 - Walking/XOR bits
16	16-bit value - Walking pattern 0x0000-0xFFFF
17	16-bit value - Walking pattern 0x0000-0xFFFF-0x0000
18	32-bit value - Fibre Channel Low Frequency Alternating
19	32-bit value - Fibre Channel Low Frequency Fixed #1
20	32-bit value - Fibre Channel Low Frequency Fixed #2
21	32-bit value - Fibre Channel Low Transition Alternating
22	32-bit value - Fibre Channel Low Transition Fixed #1
23	32-bit value - Fibre Channel Low Transition Fixed #2
24	32-bit value - Fibre Channel High Transition Fixed
25	32-bit value - Fibre Channel CJPAT - All Transition
26	32-bit value - Fibre Channel CSPAT - Supply Noise Test
27	32-bit value - Fibre Channel JTPAT - Jitter Tolerance
28	64-bit value - Alternating Blink 00000000FFFFFFFF FFFFFFFF0000000
29	64-bit value - Walking Bits 0101010101010101 0202020202020202 etc.
30	32-bit value - Network noise pattern (0x492/0x6db reversed)

No.	Description
31	16-bit value - Checkerboard
32	32-bit value - Walking pattern - repeats per FOP
33	32-bit value - Walking pattern - repeats after 4GB written
34	32-bit value - 10G Continuous Jitter Pattern
35	8-bit value - Random
36	32-byte value - Chipset Noise Pattern #1 - Modified I18
37	32-bit value - Fibre Channel Low Frequency Transitions
38	32-bit value - Fibre Channel Neutral Noise pattern #1
39	32-bit value - Fibre Channel Neutral Noise pattern #2
40	32-bit value - Fibre Channel Blink pattern #1
41	64-bit value - Bit increment/decrement
42	16-bit value - Alternating blink (bit, nibble, byte, word)
43	64-byte value - Fibre Channel ISI Killer Pattern
44	32-bit value - Fibre Channel 1KJPAT (CJTPAT plus ISI Killer)
45	8-bit value - Walking Flip/Flop Bytes 00FF0001FE0102FD02 etc.
46	32-bit value - Data pattern set to buffer memory addresses.
47	Scrambler pattern one (58 bits random, 134 bits 0.)
48	Scrambler pattern two (single bit on, then off bits with -L#.)
49	Scrambler pattern three (Off bits with -L#, then single bit on.)
50	32-bit value - Fibre Channel Blink pattern #2
51	32-bit value - Fibre Channel Blink pattern #3 (high frequency)
52	32-bit value - Fibre Channel Blink pattern #4 (2 bits off/on)
53	32-bit value - Fibre Channel Blink pattern #5 (low frequency)
54	32-bit value - SATA Composite-bit Pattern (COMP)
55	32-bit value - SATA Low Translation Density Pattern (LTDP)
56	32-bit value - SATA High Translation Density Pattern (HTDP)
57	32-bit value - SATA Low Frequency Spectral Content Pattern (LFSCP)
58	32-bit value - SATA Simultaneous Switching Outputs Pattern (SSOP)
59	32-bit value - SATA Lone-Bit Pattern (LBP)
60	8-bit value - 8b/10b Random Neutral Pattern
61	8-bit value - SAS CJTPAT (JTPAT RD+/RD-)
62	8-bit value - 8b/10b Random Neutral Pattern with inversion
63	Fibre Channel JSPAT - scrambled jitter pattern
64	Fibre Channel JTSPAT - jitter tolerance scrambled pattern
69	8-bit value - Fixed pattern of 0x00000000

No.	Description
94	32-bit value - PCI Parity alternating pattern
96	16-bit value - Alternating 0xAAAA, 0x5555
99	Custom Blink (use -L# to specify blink length in bits)
	l99 blink only, no walking bit
	l99w adds bit walk to both “off” and “on” cycles
	l99o adds bit walk to “on” cycle only,
	l99f adds bit walk on “off” cycle only.
	Combine with -e# (length of blinking), -E# (Data pattern hold time before transition), and -F (flip/flop pattern mode) for other interesting variations.

Appendix B
Test Guidelines and Examples

In this appendix:

This appendix shows examples of Medusa Labs Test Tools (MLTT) used in common test scenarios.

A Word About Hardware Configurations

The hardware required to drive data rates at the maximum bandwidth (wire speed) varies with the architecture that is being tested. For example, in order to test a 2 Gb/s Fibre Channel device, an initiator system running MLTT needs at least a PCI 64-bit/66 MHz slot to drive the full duplex capabilities of the link (400MB/s). It is important to consider the bandwidth capabilities of the device under test and evaluate each component of the configuration to identify any weak links.

Generally speaking, when testing for maximum bandwidth we suggest you have the fastest initiator system possible—that is, the processor, host bus, PCI, etc. A large memory capacity is also crucial to efficiently handle large I/O buffers.

Maximum Bandwidth Stress Testing

This section contains some guidelines for testing a device at the maximum bandwidth supported, for verification of capability, signal integrity testing, and/or data integrity.

In order transfer the maximum amount of data with the fewest interrupts in the host system, we recommend using large block sizes whenever possible. A system with a fast host bus and ample memory should be able to efficiently use block sizes of 512k or more. On target devices with caching capabilities, it is desirable to keep the file size small.

On enterprise class multi-processor systems, Pain is a good choice for driving wire speed traffic. Because each thread uses its own file or device offset, a large block I/O size with a file size equal to the block size is often a good choice when testing for bandwidth with Pain. This allows the best full duplex opportunities with threaded I/O.

Examples Using Pain:

Fastest possible, no data comparison:

```
pain -f\\.\physicaldrive1 -o -t8 -b512k 512k -u -n
```

Where:

`-o` = Hold target open (performance gain)

`-t8` = Create 8 worker threads

`-b512k` = 512KB block (buffer size)

`512k` = 512KB file size or offset size used by each thread

`-u` = Disable I/O signatures (slight performance gain)

`-n` = Disable data compares (great performance gain)

The following command line is the same as the prior example, with full data comparison added. Fastest possible with data comparison:

```
pain -f\\.\physicaldrive1 -o -t8 -b512k 512k
```

The Maim tool also has good full duplex capabilities in certain modes, using a single worker thread. Maim uses a single file and, for best results, we recommend that the file size be equal in size to the block size times the queue depth.

Examples Using Maim:

Fastest possible, no data comparison:

```
maim -f\\.\physicaldrive1 -m16 -o -Q16 -b512k 8 -u -n
```

Where:

`-m16` = Static queue depth (higher full duplex opportunity)

`-o` = Hold target open (performance gain)

`-Q16` = Queue depth of 16 I/Os of specified buffer size

`-b512k` = 512KB block (buffer size)

`8` = 8MB file size or offset size (total used by single worker thread)

`-u` = Disable I/O signatures (slight performance gain)

`-n` = Disable data compares (great performance gain)

The following command line is the same as the prior example, with full data comparison added. Fastest possible with data comparison:

```
maim -f\\.\physicaldrive1 -m16 -o -Q16 -b512k 8
```

Performance Testing

The approach to performance testing is similar to maximum bandwidth testing. Again, large blocks to small files at fairly low queue depths typically bring out the highest throughput levels. For IOPS tests, the queue depth will probably need to be raised as block size is lowered. Maim might get better results than Pain in some cases, as the queue depth can be raised considerably, without the overhead of high thread numbers.

Depending on the specific device or system to be tested, there are certain other variations you might want to use. For example, when testing a storage device it is interesting to run performance tests with file sizes that fit within device cache and some that overrun the cache size. This methodology gives a broader picture of the device's capabilities. As another example, when testing an intermediary device such as a switch, it is desirable to run performance tests that include various initiators to target configurations. These include one-to-one, one-to-many, many-to-one, and many-to-many.

For performance tests, data comparisons and I/O signatures should always be disabled. It is best to test with a variety of block sizes, to identify any problem areas.

High Bandwidth Example:

```
pain -f\\.\physicaldrive1 -o -t8 -b512k 512k -u -n
```

Where:

`-o` = Hold target open (performance gain)

`-t8` = Create 8 worker threads

`-b512k = 512KB block (buffer size)`

`512k` = 512KB file size or offset size used by each thread

`-u` = Disable I/O signatures (slight performance gain)

`-n` = Disable data compares (great performance gain)

High IOPS Example:

The following command line is the same as the prior example, with a smaller (512 byte) block and file size.

```
pain -f\\.\physicaldrive1 -o -t8 -b512 512 -u -n
```

The following command line uses maim as an IOPS test example, with a higher queue depth and static queue depth.

```
maim -f\\.\physicaldrive1 -o -Q32 -b512 512 -u -n -m16
```

The above examples are full-duplex-style tests. It is always a good idea to run half duplex tests (write only and read only.) To do this, use the `-w` or `-r` switches with command lines similar to those given in the examples.

When testing devices that support data compression, it is interesting to test performance with both compressible and non-compressible patterns (ex. All zeros with `-l69` and random data with `-l35`, respectively.)

Data Integrity Testing

Data integrity testing should be as comprehensive as time allows. Ideally, a wide variety of block sizes, file sizes, and data patterns should be utilized. Larger file sizes are often used to get longer streams of write or read traffic. Data patterns that are particularly aggravating to the architecture under test should be emphasized (ex. Fibre Channel, Parallel SCSI, PCI, etc.)

Many data corruption issues are discovered in association with fault injection tests. Data integrity testing should be an interactive process on devices with fault tolerant features.

Data integrity testing is also a good candidate for scripted testing. Typically, once a particular catalyst is introduced, data corruptions will manifest quickly in a test. Using scripts of short test sequences of continuously changing I/O parameters and data patterns, it is possible to achieve broad coverage in a relatively short time frame. See "Scripting" on page 233, in Chapter 7, "Catapult Test Tool Automation" for more information on setting up scripted tests.

Whenever possible, you should use protocol analyzers for data integrity testing. You can set the analyzer to trigger on a special value with MLTT `-!` or `-#` switches. Even when analyzers are not available, you should use the `-!` or `-#` switches, as they also cause a complete dump of the write and read buffers contents to files. This is extremely useful when debugging a corruption.

Examples:

Pain with a standard PCI aggravating pattern:

```
pain -f\\.\physicaldrive1 -o -t10 -b512k 10 -l14 -!
```

Maim with a standard Fibre Channel aggravating pattern:

```
maim -f\\.\physicaldrive1 -o -Q8 -b512k 100 -l25 -!
```

Backup or Snapshot Testing

MLTT has a data verification mode that works well for situations involving the validation of backup or snapshot implementations. You use the `-V` switch for this purpose; it is available in both Pain and Maim. This switch verifies the data in a file or on a device against a specified data pattern. The data can exist in a different location than where it was originally written.

In order for this switch to work, you must specify the exact data arguments used to create the data. The data should be created in a single write pass without I/O signatures. (See Appendix D, "I/O Signatures" for more information about this topic.)

Example:

One write pass of a pattern, with no signatures:

```
maim -Q16 -b128k -l17 -L4 -u -w 100 -i1 -fg:\data1\test.dat
```

Verification of pattern in file at another location:

```
maim -Q16 -b128k -l17 -L4 -u 100 -V -fh:\data2\test.dat
```

Maximum Queue Testing

A good target test case is its handling of a queue full condition. The point at which this can happen varies by target, but in general, most targets will handle no more than 256 concurrent requests. Maim's asynchronous I/O dispatching is an excellent method for testing queue full conditions.

Example:

The following command line launches Maim with a queue depth burst of 257 I/Os. Note that the file size specified must be large enough to contain the indicated block size times the queue depth.

```
maim -f\\.\physicaldrive1 -o -b64k 20 -Q257
```

Full Coverage Target Testing

Maim has a full coverage I/O mode that you can use in tests that cover the full capacity of a target's storage space. This mode runs across the entire device, with writes and reads operating in sections of the device equal to the file size at a time.

Example:

The following command lines run write and read commands, with data comparison, across the entire drive in 50MB sections at a time.

```
maim -f\\.\physicaldrive1 -O -b64k 50 -Q16 -m18
```

```
maim -f\\.\physicaldrive1 -O -b64k 50 -Q16 --full-device
```

The following command line runs a write command followed by immediate read-back with data comparison to random locations within the entire drive:

```
maim -f\\.\physicaldrive1 -O -b64k -Q16 -m17
```

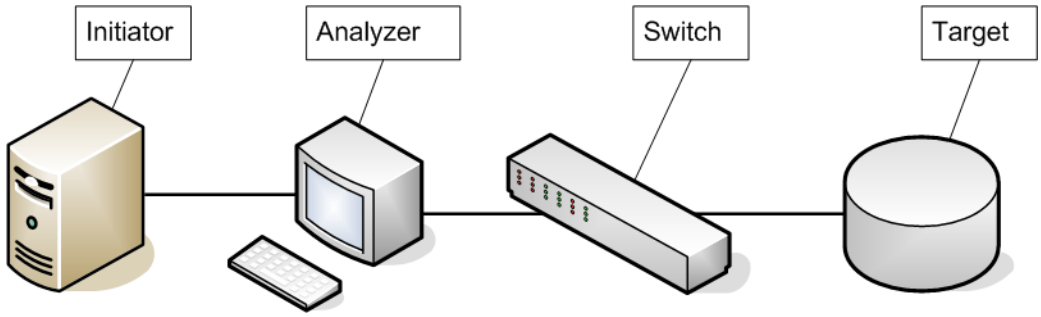
Appendix C
Debug Example

In this appendix:

This appendix contains a sample debug of a data corruption scenario.

In this example, the debug output of Medusa Labs Test Tools (MLTT), in conjunction with a protocol analyzer, can quickly isolate the device that caused a data corruption to occur. For this scenario, we use a simple Fibre Channel configuration, with a JDSU Xgig Analyzer between the HBA and a switch.

Figure 68: Fibre Channel Debug Test Configuration

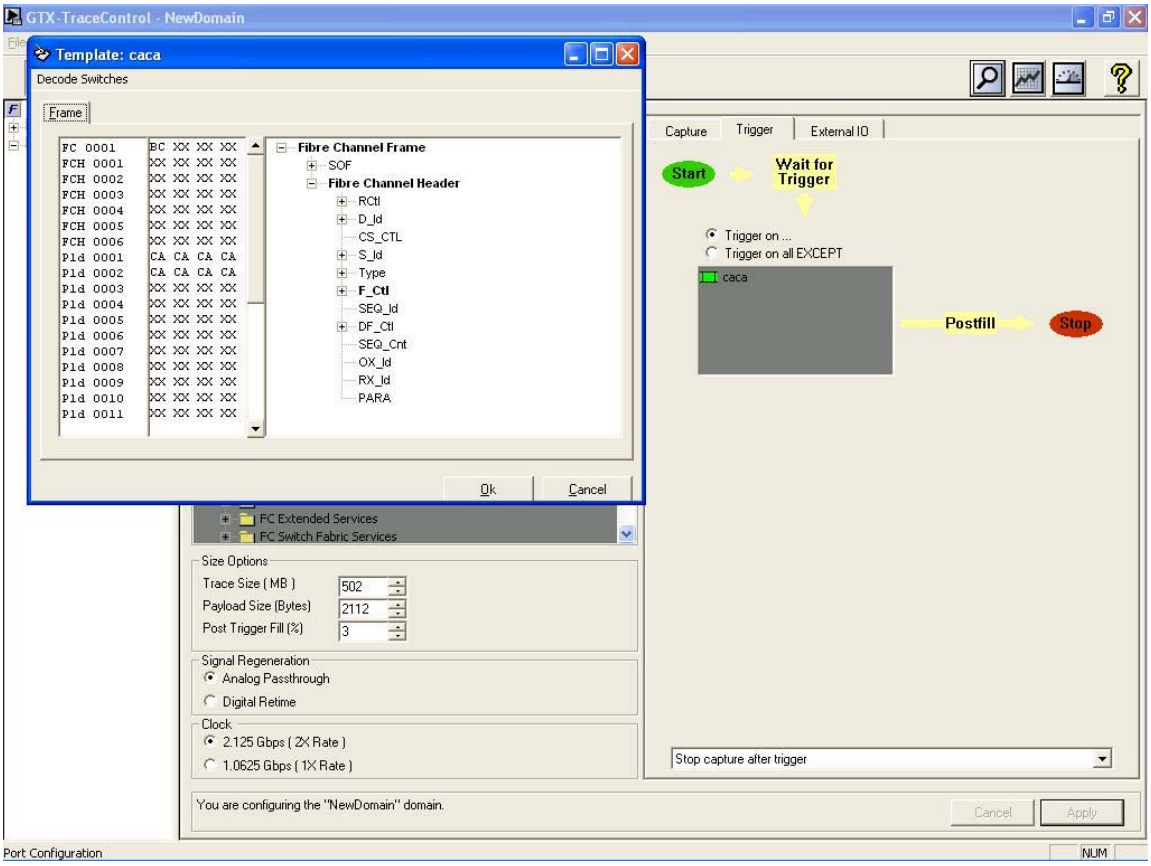


In this configuration, a basic Pain stress test is run on a Windows initiator to a physical drive on the target device. The analyzer is set to trigger on a special value sent by MLTT when the `-!` or `-#` switch is used. The trace buffer should be set to allow sufficient room for capture of relevant traffic preceding a trigger. Typically, a 90/10 split (10% post fill after trigger) is sufficient. It may be desirable to capture a smaller frame payload in some cases, but it is generally best to capture full payload when data integrity testing is performed.

Default Trigger Value

The default trigger value for a data corruption error is `0xCACACACA`. The analyzer is set to look for this value at the start of a data frame payload (Figure 69).

Figure 69: Setting the Default Trigger Value on the Analyzer



When a data corruption is detected by MLTT, the trigger I/O is immediately sent and this should be detected by the analyzer. It should be noted, however, that there are cases where the trigger I/O may be unsuccessful due to complete lack of response from a target or a device driver problem.

TRIGGER.OUT marks - for CACA trigger

If -! used, the trigger will be 0xcacacaca in p/l word 0 and 1.

If -!2 used, the trigger will be 0xdeaddead in p/l word 0 and 1.

Other values as follows:

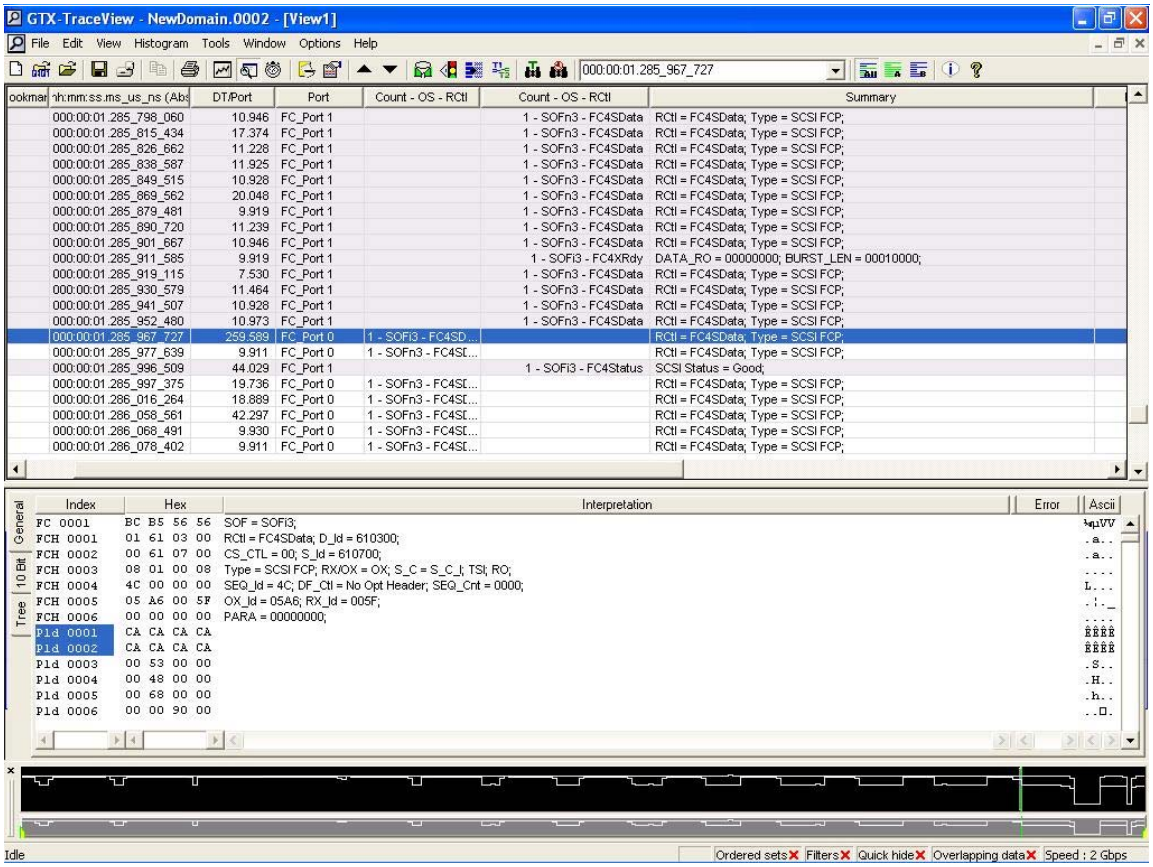
Word	Values	Conditions
2 - 7	LBA Information 2 - Corrupted LBA 3 - Starting LBA 4 - Ending LBA 5 - Starting offset (bytes) 6 - Ending offset (bytes) 7 - Base offset (bytes)	Only if -f specified on command line. Is only valid/useful if Physical Drive. If logical drive (stripe), divide by #drives and add 0x20. If -f not provided on command line, will be set to 0xCACA
8	Program Name	First four characters
9	WS_NAME	Last four characters
10	Flags Intel = DDCCBBAA Sparc = AABBCCDD	DD = thread_num CC = 01 if buffer reversed, 00 if forwards BB = 00 if I/Os forward, 01 if backwards AA = Data pattern (-l value) - in HEX
11	Loop count	
12	Buffer Size	
13	Completed I/Os	
14	Block# / Error Type Intel = DDCCBBAA Sparc = AABBCCDD	AA = Error Type (as identified in the following list) 02 (2) STARTUP_ERROR 0C (12) WRITE_ERROR 03 (3) MALLOC_ERROR 0D (13) CORRUPT_ERROR 04 (4) LOG_ERROR 0E (14) INITIAL_ERROR 05 (5) SEEK_ERROR 0F (15) REMOVE_ERROR 06 (6) RETRY_ERROR 10 (16) UNKNOWN_ERROR 07 (7) SIZE_ERROR 11 (17) TIMEOUT_ERROR 08 (8) OPEN_ERROR 12 (18) LICENSE_ERROR 09 (9) FLUSH_ERROR 13 (19) IOCTL_ERROR 0A (10) CLOSE_ERROR 14 (20) HALT_ERROR 0B (11) READ_ERROR BBCCDD = Block Number
15	Index info Intel = DDCCBBAA SPARC = AABBCCDD	Maim - AABB is pending I/Os, CCDD is I/O Index

Note that all values will be stored in AABBCCDD format (big-endian) for all platforms.

Locating the Trigger Data Frame in TraceView

To locate the corruption problem in the trace, start by finding the trigger data frame (Figure 70).

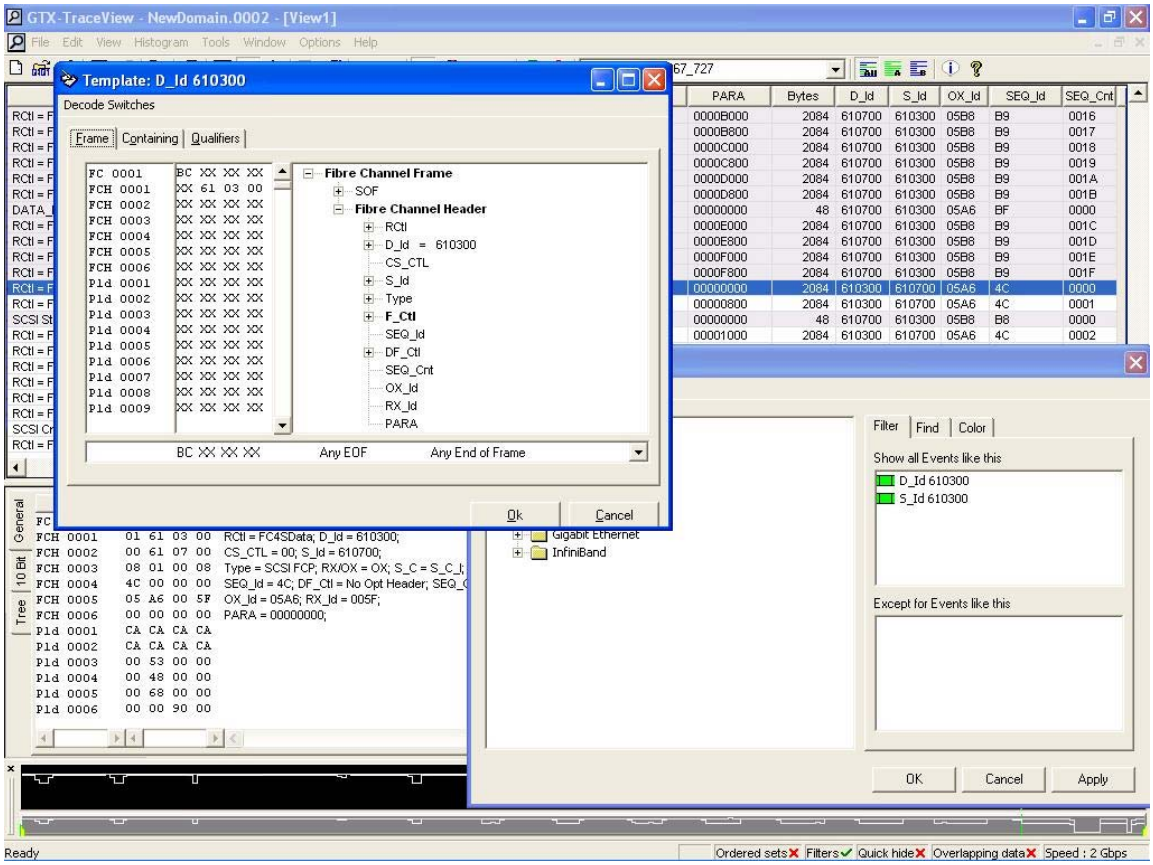
Figure 70: Locating the Trigger Data Frame in TraceView



Finding the Write and Read Operations

Once the trigger data is located, the write and read operations which resulted in the data corruption can be found in the preceding trace data. When test configurations involve multiple targets, it may be desirable to filter the trace view such that exchanges between the initiator and the target in question are displayed. In this example, the target destination ID (D_id) is used to build a pair of filters that isolate this ID as both source and destination. This allows for viewing of all bidirectional traffic between the initiator and this target.

Figure 71: Setting Filters to Isolate IDs



Error Log Created

MLTT creates an error log with the extension ".bad" when a critical error is encountered. This log is named after the thread that encountered the error in a multi-threaded test or after the host name in a single threaded test. The log file contains information when data corruptions are discovered, including a side by side listing of expected and miscompared data. A summary section includes relevant details about the corruption. Figure 72 shows the error log for this sample.

Figure 72: Sample Error Log

```
Miscompare: Offset: 0X008010, Wrote: 40130000, Read: 00000000 - returning error!
Writing trigger to offset: 0X0F0000, LBA: 0X000780
Dumping r/w buffers to: MEDUSA-00160000.3w and MEDUSA -00160000.3r...
04/15/04 15:18:22: Data corruption! Elapsed time: 00:00:01:01
Miscompare at 0X160000 bytes read in loop 312!
Device: \\.\physicaldrive1
Offset:    Returned:            Expected:
-------------------------------------------------
0X168010:  00000000 =     40130000 = @
0X168014:  00000000 =     400B400A = @ @

ERR: ----------------------------------------------------------------
ERR: Session ID:           0X000DD4 / 3540
ERR: Loop count:           312
ERR: Elapsed time:         00:00:01:01
ERR: File name:            \\.\physicaldrive1
ERR: Starting Offset:      0X900000 / 9437184
ERR: Ending Offset:        0XD00000 / 13631488
ERR: Base Offset:          0X100000 / 1048576 (already included)
ERR: * Note LBA values are valid only for physical device access.
ERR: Corrupt LBA:          0X005300
ERR: Physical LBA Range:   0X004800 to 0X006800
ERR: Data pattern:         17 / 16-bit inc/dec pattern
ERR: Pattern Cycle length: 1
ERR: Pattern Direction:    Up/Even
ERR: I/O Direction:        Backward
ERR: Write Buffer address: 0X1300000 / 19922944
ERR: Read Buffer address:  0X1310000 / 19988480
ERR: I/O Size:             0X010000 / 65536
ERR: Block number:         0X000017 / 23
ERR: Block start:          0X160000 / 1441792
ERR: Error start:          0X168010 / 1474576
ERR: Error end:            0X168018 / 1474584
ERR: Error length:         0X000008 / 8
ERR: ----------------------------------------------------------------

Retrying read on corruption...
Retry didn't show corruption!
```

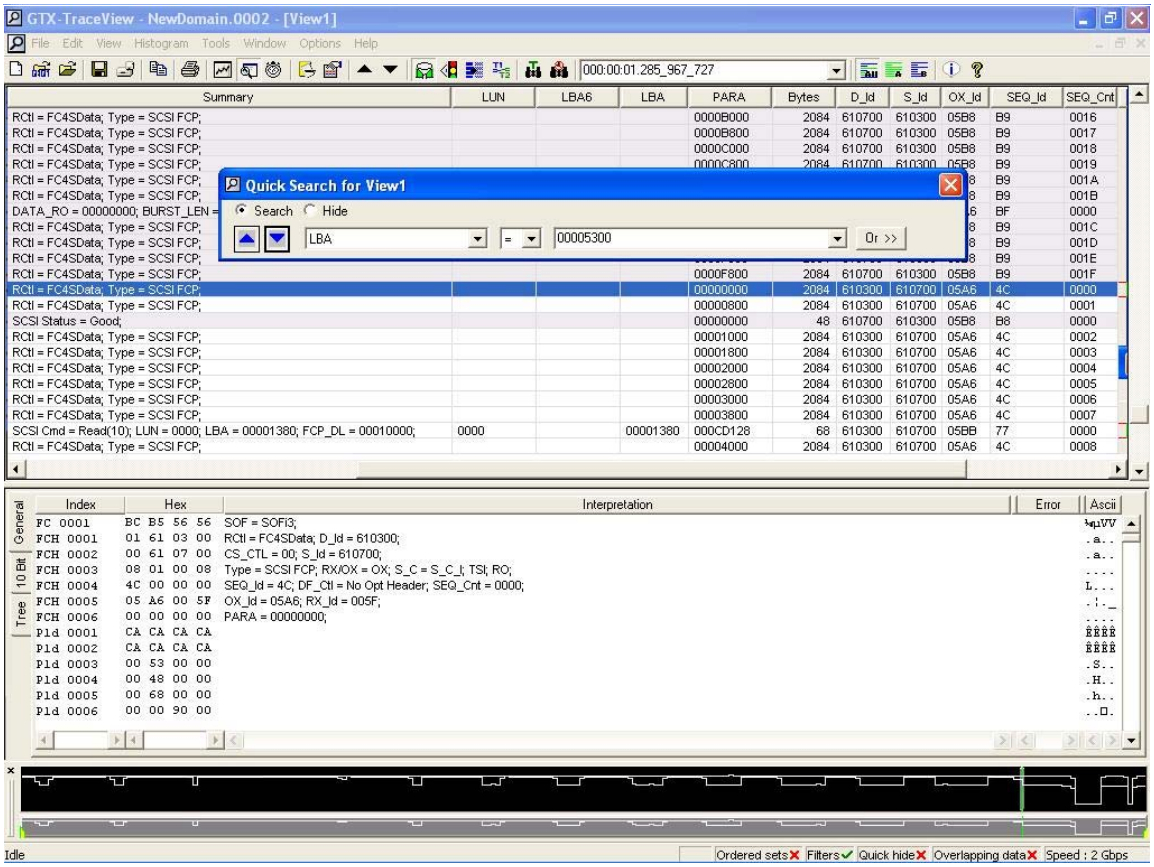
Because this example is using a physical drive as the target, the corrupt LBA reported in the error log is accurate and can be used to locate the write and read commands in the trace. Note that the corrupt LBA refers to the start of the I/O request and not necessarily the start of the data corruption within the I/O data.

In this example, the corrupt LBA is 0x005300:

```
ERR: Corrupt LBA:           0X005300
```

The last command to this LBA on the target in question is located in a backward search from the trigger point (Figure 73).

Figure 73: Backward Search for Last Command to the LBA



The trigger I/O was sent immediately upon discovery of the data corruption and the last command to the corrupt LBA before the trigger was the read command. When we find the read command, we can trace the read data to find the point where the miscompare begins. We can determine the location of the corrupt data in the data transfer from the error log by looking at the block start and error start addresses:

```
ERR: Block start:           0X160000 / 1441792
ERR: Error start:           0X168010 / 1474576
```

Subtracting the error start from the block start provides the offset of the miscompare from the start of the read data transfer:

```
 0X168010
-0X160000
 0X008010 = 32784 bytes
```

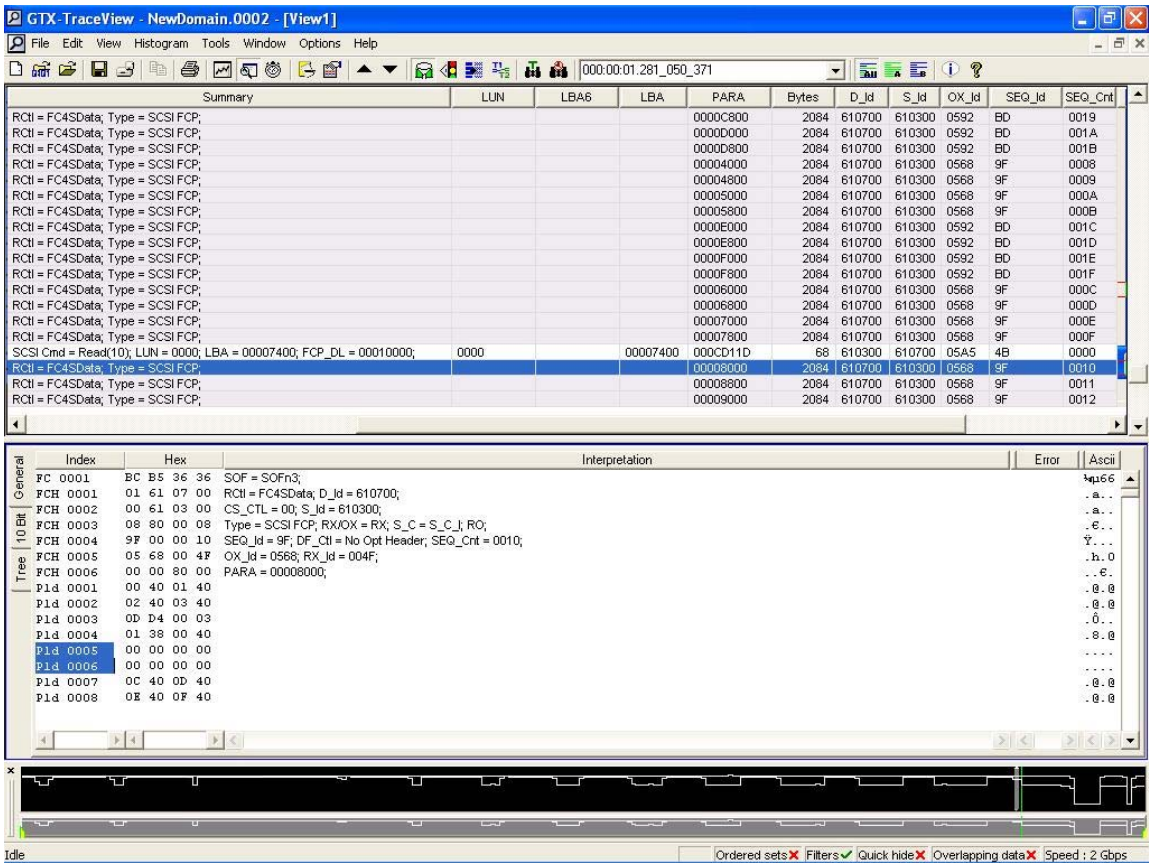
Finding the Corrupt Data Frame

Since this example is over Fibre Channel, we will divide the byte offset by the size of the frame data payload (2048 in this case) in order to locate the data frame in question.

`32784 / 2048 = 16` (frame number)

This data frame is located in the trace (frame 16 would be SEQ_Cnt 0x10). See (Figure 74):

Figure 74: Data Frame in Trace



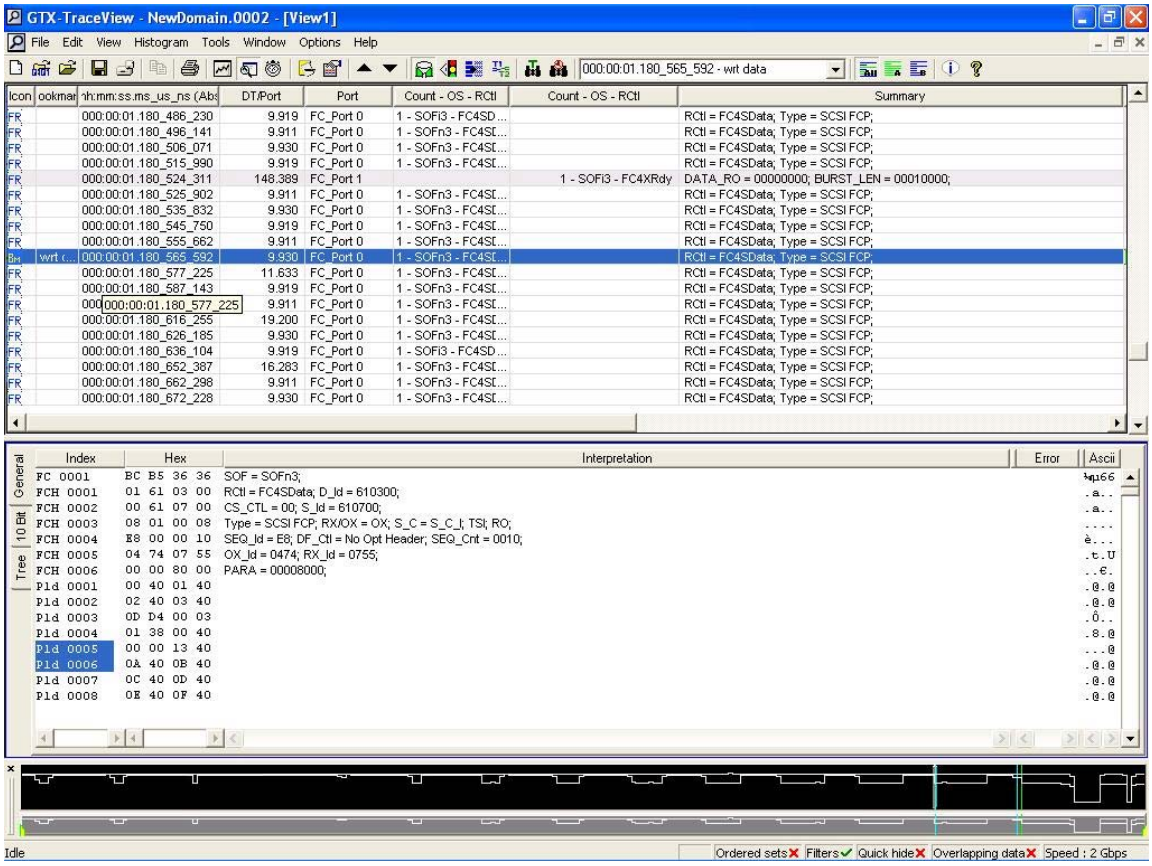
In this data frame, we find the corrupted data listed in the error log – two words of zeroes instead of the expected data.

```
Offset:     Returned:             Expected:
-------------------------------------------------
0X168010:   00000000 =      40130000 = @
0X168014:   00000000 =      400B400A = @ @
```

It is possible that the data in the trace could read correctly, indicating the corruption originated from something inside the initiator system (such as the HBA or PCI bridge) rather than the target.

Searching back for the corrupt LBA past the read command will take us back to the write command, where the expected data can be verified (Figure 75).

Figure 75: Verifying the Expected Data



It is entirely possible for the write data to be corrupted, if the corruption is due to a component in the initiator system. This would be classified as a “write corruption.” In this example, the error log shows that we are dealing with a read corruption:

```
Retrying read on corruption...
Retry didn't show corruption!
```

MLTT will always attempt a read retry when a corruption is discovered. This provides an important data point by determining whether the corruption is persistent (committed to the media) or intermittent (possibly the result of a cache error on a target.) In this example, it would appear that the target is the culprit, given that incorrect data was returned in a valid data frame with no CRC errors. It is also possible, although less likely, that the switch between the analyzer and the target caused the error. To be certain, you would want to move the analyzer to the connection between the switch and the target.

In this example, due to the nature of our configuration and the Fibre Channel protocol, we were able to easily locate the relevant commands with the LBA. It often becomes necessary to use another search method when the LBA provided by the error logs is not valid because of a file system or logical volume configuration. Also, some protocols, such as iSCSI can be difficult to debug due to commands being imbedded in packets.

In such cases, it may be necessary to search for the unique I/O signature in the block in question. Using the block and error starting addresses from the error log, it is possible to locate the I/O signature closest to the corruption. (Refer to Appendix D, “I/O Signatures” for more information) In our current example, we know that the error occurred at offset 0X8010 into the I/O.

```
ERR: Block start:          0X160000 / 1441792
ERR: Error start:          0X168010 / 1474576

 0X168010
-0X160000
 0X008010
```

Using a hex editor, the write or read buffer data can be opened and this offset may be located (Figure 76). MLTT automatically dumps the write and read buffers to files with the `-!` or `-#` switches are used. The file names are provided in the error log.

Figure 76: Locating the Offset

```
Miscompare: Offset: 0X008010, Wrote: 40130000, Read: 00000000 - returning error!
Writing trigger to offset: 0X0F0000, LBA: 0X000780
Dumping r/w buffers to: MEDUSA-00160000.3w and MEDUSA -00160000.3r...
```

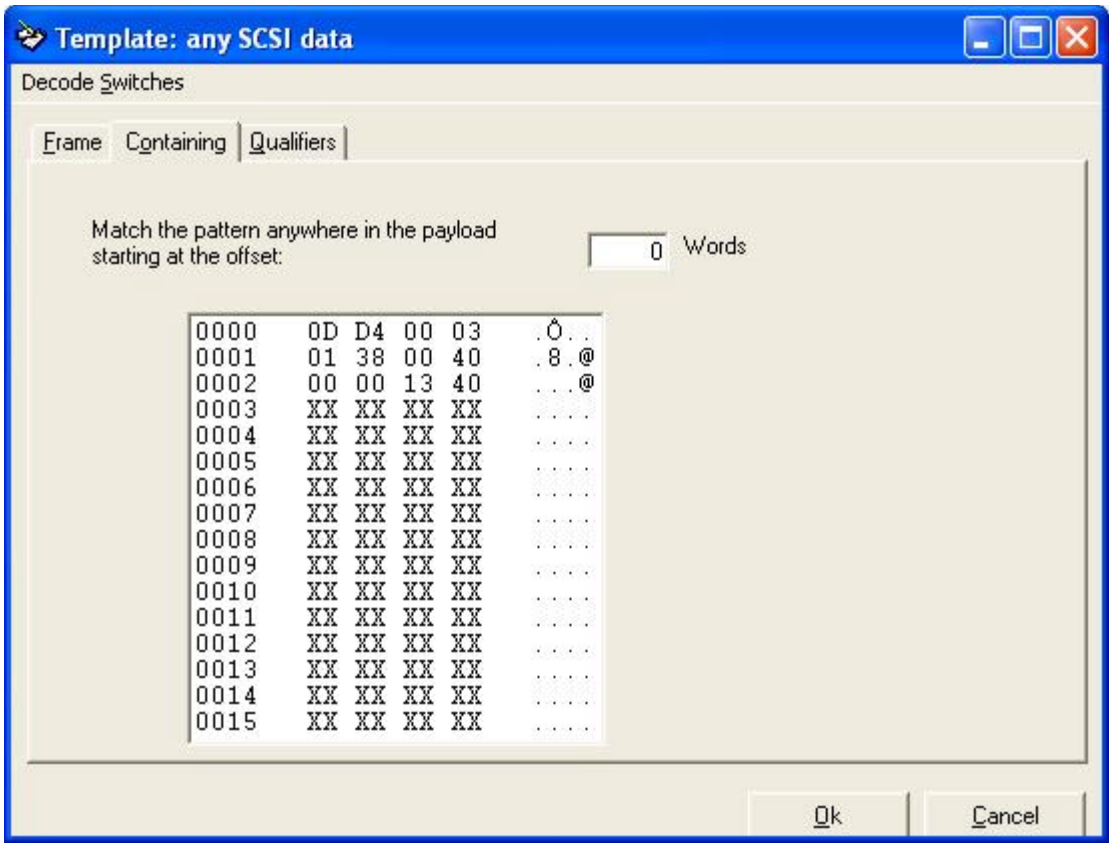
Using I/O Signatures

The I/O signatures occur every sector size, which is typically 512 bytes or 0x200 hex. The signatures are three words in length and are placed at a two word offset into each sector area. In this example, the nearest signature is at 0x8008 in the buffer files.

```
0X008000   00 40 01 40      Sector Start
0X008004   02 40 03 40
0X008008   0D D4 00 03      Signature Start
0X00800C   01 38 00 40
0X008010   00 00 13 40      Signature End
0X008014   0A 40 0B 40
0X008018   0C 40 0D 40
0X00801C   0E 40 0F 40
0X008020   10 40 11 40
0X008024   12 40 13 40
```

This signature can be used to set up a search filter in a trace viewer application (Figure 77).

Figure 77: Using I/O Signature for a Search Filter in TraceView



Using the FindLBA Utility

FindLBA is useful in cases where the logical block address (LBA) reported in the I/O tool error logs is not accurate because the tools are not directly referencing areas of the physical media. You can use FindLBA in conjunction with a protocol analyzer to identify the actual LBA corresponding to a file offset reported by the test tools. FindLBA sends a “ping” of consecutive reads to a specified offset, which you can identify in a protocol trace. FindLBA is most useful when you need help finding I/O commands that resulted in data corruption in a protocol trace capture.

Note: When entering the offset in the FindLBAcommand, enter the offset in bytes only. FindLBA does not support common modifiers, such as k, M, etc.

The following examples show how the FindLBA utility can be used.

Example 1

The thread1.bad log file from a test case that showed data corruption has the lines:

```
ERR: Error length:         0x0200 / 512”,  “Miscompare: Offset: 0x00000000000FF800
```

and

```
ERR: Starting offset:      0x100000 / 1048576
```

This means that there are 0x200 bytes of corrupted data 0x100000 + 0xFF800 or 0x1FF800 bytes into the file. On a Microsoft Windows system, you can run the command:

```
findlba -f\\.\PhysicalDrive2 -o0x1ff800
```

The Xgig trace will show SCSI Reads to that LBA on the storage where the data was detected as corrupt. You can then use the trace to find the actual LBAs that were reported as corrupt.

Example 2

You want to send a 16,384 byte Read to LBA 0x1000 on a disk drive formatted for 512 bytes per sector. The Offset is 0x200 bytes/sector times LBA 0x1000. The buffer size is 0x4000.

On a Microsoft Windows system, the command is:

```
findlba -f\\.\PhysicalDrive2 -o0x200000 -b0x4000
```

Appendix D
I/O Signatures

By default, Medusa Labs Test Tools (MLTT) adds a unique mark or signature to each I/O. This signature is placed in the selected data pattern at every sector interval in the I/O buffer. The purpose of this signature is two-fold.

First, this is a vital component of data integrity checking in MLTT. A constantly changing data stream is essential for catching cases of out-of-order write completions or “stale data” corruptions. Stale data corruptions are cases where data from an old write operation is returned after a new write operation and read back with data comparison to the same LBA. If the exact same data were used for every write operation, there would be no way to detect if the read data returned is for the immediately preceding write.

Second, I/O signatures are extremely useful in debugging issues discovered with MLTT. The information embedded in the signature can be correlated with log files from a test run to determine the initiator, target, I/O position within a file, LBA, etc.

The signature used by MLTT has the following format.

```
Offset    Data
0x0008:  AAAABBBB
0x000C:  CCDDDDDD
0x0010:  EEEEEEEE
```

Where:

`AAAA` = Session ID (A unique ID created from the initiator and target names.)

`BBBB` = Thread number (Pain) or I/O Index number (Maim)

`CC` = Loop count (Based on file operations (FOPS))

`DDDDDD` = Block number

`EEEEEEEE` = LBA (Only accurate on physical devices.)

When the time stamp switch (`-U`) is used, the following addition is made to the I/O signature:

```
Offset    Data

0x0014:  FFFFFFFF
```

Where:

`FFFFFFFF` = Standard 32-bit OS time stamp in seconds

Additionally, millisecond resolution may be added with the `-Um` switch. This is a two byte addition to the time stamp in the I/O signature:

```
Offset    Data

0x0018:  GGGGXXXX
```

Where:

`GGGG` = Millisecond time stamp

`XXXX` = Data pattern

This signature is inserted into the data pattern at every sector interval (normally every 512 bytes.)

Example (with -Um switch and data pattern of all 0xAA):

```
00000000  AA AA AA AA
00000004  AA AA AA AA
00000008  04 E9 00 01
0000000C  00 00 00 00
00000010  00 00 08 00
00000014  40 86 D1 7B
00000018  01 A9 AA AA
0000001C  AA AA AA AA
00000020  AA AA AA AA
…
00000200  AA AA AA AA
00000204  AA AA AA AA
00000208  04 E9 00 01
0000020C  00 00 00 00
00000210  00 00 08 01
00000214  40 86 D1 7B
00000218  01 A9 AA AA
0000021C  AA AA AA AA
00000220  AA AA AA AA
…etc.
```

Note that the Session ID is written to the error logs, so you can correlate the logs with traces or captures. When running tests to file systems or logical partitions, you can use the LBA to determine relative file offset by simply multiplying the LBA by the sector size.

Appendix E
Exit Codes

The Medusa Labs Test Tools (MLTT) return codes at exit that you can use in custom batch files or shell scripts to take action when an error occurs during a scripted run.

The following example shows an exit code in a script:

```
echo off
REM Run pain for 5 minutes
pain -o -l35 -t4 -d300
REM Check exit code, 2 or higher means there was an error
If ERRORLEVEL 2 goto ERROR
echo Success! Exit code: %ERRORLEVEL%
goto END
:ERROR
echo Error! Exit code: %ERRORLEVEL%
:END
```

The exit codes used by MLTT are listed below.

```
0 SUCCESS
1 LOOP_DONE
2 STARTUP_ERROR
3 MALLOC_ERROR
4 LOG_ERROR
5 SEEK_ERROR
6 RETRY_ERROR
7 SIZE_ERROR
8 OPEN_ERROR
9 FLUSH_ERROR
10 CLOSE_ERROR
11 READ_ERROR
12 WRITE_ERROR
13 CORRUPT_ERROR
14 INITIAL_ERROR
15 REMOVE_ERROR
16 UNKNOWN_ERROR
17 TIMEOUT_ERROR
18 LICENSE_ERROR
19 IOCTL_ERROR
20 HALT_ERROR
```

Each exit code is described in the following section.

Exit Code Descriptions

The exit code descriptions are listed in numerical order.

SUCCESS (0)

This value is used as a generic non-error exit code. It is normally only returned when the Test Tool exits from a display of the on-line help.

LOOP_DONE (1)

This value indicates a normal exit from a test that was limited by iteration count (`-i` switch) or duration (`-d` switch.)

STARTUP_ERROR (2)

This value indicates an error was encountered during the processing of environment variables or command line switches.

MALLOC_ERROR (3)

This value indicates there was an error encountered in a memory allocation attempt.

LOG_ERROR (4)

This value indicates an error was encountered while accessing one of the log files.

SEEK_ERROR (5)

This value indicates a seek operation on a target device or file has failed.

RETRY_ERROR (6)

This value indicates an I/O operation has failed and attempts to retry the operation were unsuccessful.

SIZE_ERROR (7)

This value indicates an I/O operation has failed due to an unexpected number of bytes returned or an unexpected end of file was reached. For example, this error would occur if a read of 64KB returned only 62KB.

OPEN_ERROR (8)

This value indicates an error occurred on an attempt to open a target device or file during a test run.

FLUSH_ERROR (9)

This value indicates an error occurred on an attempt to flush data from a write operation from cache to the target device or file.

CLOSE_ERROR (10)

This value indicates that an attempt to close an open handle to a target device or file has failed.

READ_ERROR (11)

This value indicates a read operation on a target device or file has failed.

WRITE_ERROR (12)

This value indicates a write operation on a target device or file has failed.

CORRUPT_ERROR (13)

This value indicates data corruption has been detected on a target device or file.

INITIAL_ERROR (14)

This value indicates the initial open of a target device or file has failed.

REMOVE_ERROR (15)

This value is not currently used in MLTT. Device open, read, or write errors due to device removal will be reported specifically.

UNKNOWN_ERROR (16)

This value indicates an error has occurred that is not classifiable as any other specific defined error condition.

TIMEOUT_ERROR (17)

This value indicates that one or more individual I/O operations have not completed within the monitoring time period (5 seconds by default.)

LICENSE_ERROR (18)

This value indicates an error has occurred during a security check against the Test Tool’s license. This may be because a license has expired or the license file could not be located.

IOCTL_ERROR (19)

This value indicates an error has occurred during an IOCTL operation. Some MLTT use low level IOCTL calls for various operations on target devices.

HALT_ERROR (20)

This value indicates that ALL I/O traffic has ceased during the monitoring time period (5 seconds by default.)

Appendix F
Architecture Bandwidths

This appendix lists bandwidth capabilities for various architectures.

PCI

32bit/33 MHz - 132 MB/sec

64bit/33 MHz - 264 MB/sec

64bit/66 MHz - 528 MB/sec

PCI-X

64bit/100 MHz - 792 MB/sec

64bit/133 MHz - 1 GB/sec

PCI-Express

1x=500 MB/s (Full Duplex)

4x=2 GB/s (Full Duplex)

8x=4 GB/s (Full Duplex)

16x=8 GB/s (Full Duplex)

Fibre Channel (Full Duplex)

1Gb - 200 MB/sec

2Gb - 400 MB/sec

4Gb - 800 MB/sec

8Gb - 1600 MB/sec

10Gb - 2.4 GB/sec

16Gb - 3200 MB/sec

Fast Ethernet (Full Duplex)

25 MB/sec

Gigabit Ethernet (Full Duplex)

250 MB/sec

10Gb - 2500 MB/s

SAS

1.5Gb - 150 MB/s

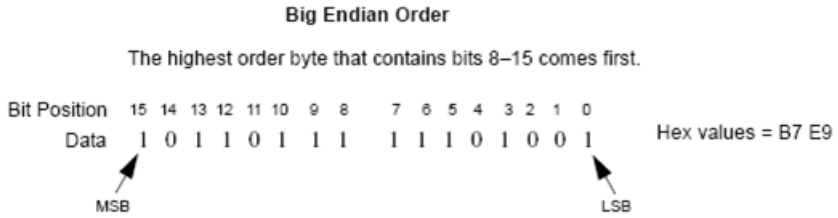
3.0Gb - 300 MB/s

6.0Gb - 600 MB/s

Glossary

Big Endian

Big Endian is the order in which bytes are arranged. When using multiple bytes, you must identify their order. This is similar to identifying the order of bits by LSB and MSB. The term big endian is a term used by designers of computer processors to identify data in the following order.



Block

A unit equivalent to the specified buffer (I/O) size. This term is used to describe the I/Os used to run the length of the specified file size. For example, a 1MB file would be comprised of 16 total 64k blocks.

Buffer Size (I/O Size)

The size (length) in bytes of the data sequence to be written or read in an operation. This is the size requested at the application level and does not necessarily correlate to the actual transfer size sent to the target device. The operating system or an underlying device driver will likely break a large I/O size down into several smaller I/O transactions.

Data Pattern

The payload sent to the target in each I/O operation. Data patterns are typically a sequence of raw data that result in signal aggravation of specific system components, bus, or serial architectures.

File Size

Used in a couple of different ways in the context of MLTT. When the tools are used on a file system, file size refers to the size of the file used by each worker (thread.) When the tools are used on a physical or logical device, file size refers to the extent of space accessed by each worker on the device.

Flag

A flag generically refers to an option passed to an I/O operation (for example, the cache options available on the -R and -W switches.)

FOP

A file operation is a complete write and read pass through a file or device extent of the specified file size.

Full-Device Coverage

An I/O test that covers the full length of a target device. Maim has an I/O mode (-m18) which will cover the entire length of a target.

For full device coverage -m18, “file size” given in the command has a different meaning. Internally, it is how far each thread travels for write before rewinding and doing the reads – then move on to the next “stroke area”. The “File size = 512KB” shown in the log file is the 512KB given in the command line which is used for the “stroke size”. For non-full device modes, “file size” == “stroke size”.

“FILE SIZE:” shown in the sample header is the target device size divide by the number of threads adjusted to fit as multiple of buffer size – it is the total per-thread contiguous area to be covered, and it should correspond to the LBA range shown per thread.

Initiator

A computer system running any supported operating system and MLTT.

I/O

A single command (write or read) issued to a device. An I/O is active or pending from the time the command is issued until status is returned or the command is aborted.

LBA

Logical block address.

Miscompare (Data Corruption)

A miscompare occurs when the value of a sequence of data reads back differently than it was written. At some point in the data transfer, the data was somehow corrupted so the write data and the read data do not match.

There are two primary types of data corruptions: Write corruption and read corruption.

Write corruption is indicated by the a case where every subsequent read of the data produces the same corrupted data. This is a result of the data being committed to the media in a corrupted state. In this case, the corruption occurred at some point during the write data transfer.

Read corruption is indicated when the initial read of the data is returned in a corrupted state, but a subsequent read produces the correct data. The data was corrupted at some point during the first read. When the data is read back again, the correct data is returned because the write operation successfully committed the data to media. It is possible for a follow-up read to return corrupted data and still be a case of read corruption. In a rare case like this, the follow-up read may return data that is corrupted in a different manner from the first read.

Queue Depth	Queue depth refers to the number of I/O operations (write or read) that are pending at any given time. An I/O is considered to be pending from the time the command is sent until status is received or the command is aborted. Generically, the thread count in synchronous tools such as Pain, correlates to queue depth. In the asynchronous tools, queue depth is specified on the command line.
Random Access	Random access refers to I/O operations across a device at randomly selected offsets.
Secure Erase	The Secure Erase configuration editor erases the data on a Solid State Drive leaving it in a *clean* state after the test process using the configuration is complete.
SMART	Retrieves Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) attributes and status from target devices and logs them.
SSD	Solid State Drive
Target	Any physical or logical device that is addressable by an operating system as a storage device. Examples include a local hard drive, a network share, or a SAN storage controller. The target is specified to MLTT with the -f switch. The Pain tool also has a memory-only mode where system memory is used as the target for host bus or memory subsystem testing.
Thread	I/O workers are executed in the context of threads within the tool processes. The number of workers or threads depends on the specific tool. Pain utilizes a single I/O per thread architecture. Maim uses a single thread per target by default but more can be specified with -t switch.
Trigger	MLTT has built-in functionality for send a special write command with a data sequence that can be used to trigger a protocol analyzer. This feature is enabled with the -! or -# switches.
Trim	The SSD Trim configuration erases specified data blocks. It may be run as a target Solid State Drive pre-conditioning step before running I/O tests.
Walking Pattern	A walking pattern is a data pattern than increments or decrements a value a bit at a time, for example, 0x01, 0x02, 0x03...0xFF.

Index

Symbols

Numbers

A

B

C

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Network and Service Enablement Regional Sales

North America	Latin America	Asia Pacific	EMEA	www.jdsu.com
Toll Free: 1 855 ASK JDSU	Tel: +55 11 5503 3800	Tel: +852 2892 0990	Tel: +49 7121 86 2222	

Version 7.0
November 2014
English