# Medusa Labs Test Tools Suite

Version 7.8

User’s Guide

# Medusa Labs Test Tools Suite

## Version 7.8

User's Guide

VIAVI Solutions
1-844-GO-VIAVI
www.viavisolutions.com

## Notice

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

## Copyright/Trademarks

## Copyright release

## Terms and conditions

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

## Federal Communications Commission (FCC) Notice

This product was tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This product generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this product in a residential area is likely to cause harmful interference, in which case you will be required to correct the interference at your own expense.

The authority to operate this product is conditioned by the requirements that no modifications be made to the equipment unless the changes or modifications are expressly approved by VIAVI.

Contents

# About This Guide

This preface explains how to use this manual. Topics discussed include the following:

## Purpose and Scope

The purpose of this guide is to help you successfully use the Medusa Labs Test Tools Suite features and capabilities. This guide includes task-based instructions that describe how to use the graphical user interface (GUI) and the command line switches of the Medusa Labs Test Tools Suite.

## Assumptions

This guide is intended for novice, intermediate, and experienced users who want to use the Medusa Labs Test Tools Suite effectively and efficiently. We are assuming that you have basic computer and mouse/track ball experience and are familiar with basic telecommunication concepts and terminology.

## Related Information

This is the user's guide for the Medusa Labs Test Tools Suite. It provides basic instructions for using the Medusa Labs Test Tools Suite and contact information for VIAVI's Technical Assistance Center (TAC).

Use this guide in conjunction with the following information:

- The *Medusa Labs Test Tools Suite Installation Guide* which provides detailed instructions for installing the Medusa Labs Test Tools Suite.

## Conventions

This guide uses typographical and symbols conventions as described in the following tables.

**Table 1** Text formatting and other typographical conventions

| Item(s) | Example(s) |
|---|---|
| Buttons, keys, or switches that you press or flip on a physical device. | Press the **On** button.<br>– Press the **Enter** key.<br>– Flip the **Power** switch to the on position. |
| Buttons, links, menus, menu options, tabs, or fields on a PC-based or Web-based user interface that you click, select, or type information into. | Click **Start**.<br>– Click **File > Properties**.<br>– Click the **Properties** tab.<br>– Type the name of the probe in the **Probe Name** field. |
| Directory names, file names, and code and output messages that appear in a command line interface or in some graphical user interfaces (GUIs). | **`$NANGT_DATA_DIR/results`** (directory)<br>– **`test_products/users/defaultUser.xml`** (file name)<br>– **`All results okay.`** (output message) |
| Text you must type exactly as shown into a command line interface, text file, or a GUI text field. | – Restart the applications on the server using the following command: **`$BASEDIR/startup/npiu_init restart`**<br>Type: `a:\set.exe` in the dialog box. |
| References to guides, books, and other publications appear in *this typeface*. | Refer to *Newton's Telecom Dictionary.* |
| Command line option separators. | **`platform [a\|b\|e]`** |
| Optional arguments (text variables in code). | **`login [platform name]`** |
| Required arguments (text variables in code). | **`<password>`** |

**Table 2** Symbol conventions

| | |
|---|---|
| ▷ | This symbol indicates a note that includes important supplemental information or tips related to the main text. |
| ⚠ | This symbol represents a general hazard. It may be associated with either a DANGER, WARNING, CAUTION, or ALERT message. See Table 3 for more information. |
| ❗ | This symbol represents an alert. It indicates that there is an action that must be performed in order to protect equipment and data or to avoid software damage and service interruption. |
| ⚡ | This symbol represents hazardous voltages. It may be associated with either a DANGER, WARNING, CAUTION, or ALERT message. See Table 3 for more information. |
| 💥 | This symbol represents a risk of explosion. It may be associated with either a DANGER, WARNING, CAUTION or ALERT message. See Table 3 for more information. |
| ♨ | This symbol represents a risk of a hot surface. It may be associated with either a DANGER, WARNING, CAUTION, or ALERT message. See Table 3 for more information. |
| ✺ | This symbol represents a risk associated with fiber optic lasers. It may be associated with either a DANGER, WARNING, CAUTION or ALERT message. See Table 3 for more information. |
| 🗑 | This symbol, located on the equipment, battery, or the packaging indicates that the equipment or battery must not be disposed of in a land-fill site or as municipal waste, and should be disposed of according to your national regulations. |

**Table 3** Safety definitions

| Term | Definition |
|---|---|
| **DANGER** | Indicates a potentially hazardous situation that, if not avoided, *will* result in death or serious injury. It may be associated with either a general hazard, high voltage, or other symbol. See Table 2 for more information. |
| **WARNING** | Indicates a potentially hazardous situation that, if not avoided, *could* result in death or serious injury. It may be associated with either a general hazard, high voltage, or other symbol. See Table 2 for more information. |
| **CAUTION** | Indicates a potentially hazardous situation that, if not avoided, could result in minor or moderate injury and/or damage to equipment.<br>It may be associated with either a general hazard, high voltage, or risk of explosion symbol. See Table 2 for more information.<br>When applied to software actions, indicates a situation that, if not avoided, could result in loss of data or a disruption of software operation. |
| **ALERT** | Indicates that there is an action that must be performed in order to protect equipment and data or to avoid software damage and service interruption. |

## Technical Assistance

If you require technical assistance,
call 1-844-GO-VIAVI (1-844-468-4284) or
e-mail Techsupport-snt@viavisolutions.com.

For the latest TAC information, go to
http://www.viavisolutions.com/en/services-and-support/support/technical-assistance.

### Medusa Labs Test Tools Product Information and Assistance

For product information and download material, to go
www.viavisolutions.com/medusatools.

You can contact the Medusa Labs Technical Support directly at
techsupport-medusa@viavisolutions.com.

1

# About Medusa Labs Test Tools Suite

This chapter provides a general description of the Medusa Labs Test Tools Suite. Topics discussed in this chapter include the following:

## What's New in this Medusa Labs Test Tools Version

**The latest version of Medusa Labs Test Tools documented in this guide is Version 7.8.**
**Medusa Labs Test Tools (MLTT) Suite has the following updates for the Version 7.x releases:**

### Version 7.8

- Added support for specifying NUMA nodes as memory targets for "pain -m9" and "pain -m10" modes. An example use case is generating I/O to CXL.mem devices configured as system RAM (memory-only NUMA node).
  - For more information, see "Target Specification" on page 182
- Added capabilities for split write-read I/O channels.
  - For more information, see "-f Target" on page 182
- Added support for data comparison using non-uniform buffer sizes for both sequential and random-access I/O.
  - For more information, see "-b Buffer size" on page 184
- Enabled reverification mode (-V) for mapped random-access I/O
  - For more information, see "-Vw Write-once for Reverification" on page 276, "--random-x-map Random Access Map" on page 249, and "-V Reverify Existing Data to a Specified Data Pattern/Verify Journaled Write Operations" on page 272
- Added capabilities to specify exact area to be used on a target device
  - Refer to "--f Target" on page 251 and "--target-partition Target Partition Range" on page 255

### Version 7.7

- Fixed support for Windows Systems that contain >64 logical CPUs
  - For more information, see "-T Set I/O Thread/CPU Affinity" on page 202
- Added NUMA awareness
  - For more information, see "-T Set I/O Thread/CPU Affinity" on page 202
- Added direct-access (DAX) to Linux PMEM devices (e.g. NVDIMM, CXL.mem)
  - Normal I/O generation using pain tool
- Added support for Linux ZNS devices
  - Normal I/O generation using pain and maim tools
  - Pass-Through mode ("- -ptio") with pain tool, including the option to use "Zone Append" command
  - Added new CLI option to reset ZNS zones. For more information, see "--nvme-reset-zone ZNS Zone Reset" on page 224

### Version 7.6

- Added support for concurrent workloads and sequential workload groups.
  - For more information, see "Concurrent Workloads and Workload Groups" on page 192.
- Added support for zoned I/O distribution. An example use case is being able to run a JDEC JESD219 test case.
  - For more information, see "-% I/O Profile Specification" on page 235.
- Added support for random-access map to access unique, non-overlapping LBAs.
  - For more information, see "--random-x-map Random Access Map" on page 249.
- Added support for I/O repeat count.
  - For more information, see "--io-repeat I/O Repeat Count" on page 249.

### Version 7.5

- The ability to export test plans as Python, Bash, and Powershell scripts was added.
  - Refer to "Export Selected Test Plans..." in chapter 2 for more information.
- Switch options for playing back a trace and complementing switches were added.
  - Refer to "--io-trace-play Streaming a Trace to a Target" and "--io-trace-no-prescan Opting out of Initial Prescan" in chapter 4 for more information.
- Switch option for enabling a target file for each thread was added.
  - Refer to "--file-per-thread   Create Target Files for each Thread" in Chapter 4 for more information.

**Version 7.4**

- License Administration Enhancements were added
  - Refer to"Licensing Administration" in Chapter 1.
- I/O Operation History Trace and Payload Data Logging options were added
  - Refer to Chapter 4 Command Line Switches and Chapter 5 Logging Output for more information about these features.
- NVMe Administrative and NVM Commands were added
  - Refer to Chapter 4 Command Line Switches and Chapter 5 Logging Output for more information about these features.
- Option to Flush the In-Memory Journal Once was added
  - Refer to Chapter 4 Command Line Switches for more information about this feature.
- Option to Log File Output Timestamps in UTC was added
  - Refer to Chapter 4 Command Line Switches for more information about this feature.
- Option to add columns for Per-Sample Period Latency Histogram and Per-Sample Period Min, Max, and Average I/O Completion Times to the CSV output file was added
  - Refer to Chapter 4 Command Line Switches and Chapter 5 Logging Output for more information about this feature.
- Support of Linux on POWER was added

**Version 7.3**

- Added Medusa Labs Test Tools Basics section.
  - Refer to "MLTT Basics" in Chapter 1.
- Added an appendix for the Sock Test Tool.
  - Refer to Appendix N "Sock Test Tool" for information about using this feature.

**Version 7.2**

- Support of Microsoft Windows Server 2016 Datacenter Edition and Standard Edition
  - Refer to the *Medusa Labs Test Tools Suite Version 7.2 Installation Guide*.
- Sample Test Plans were added
  - Refer to Chapter 2 "Using the Graphical User Interface" for information about using this feature.
- Custom Graphing features providing X-axis options were added
  - Refer to Chapter 2 "Using the Graphical User Interface" for information about using this feature.
- TCP Incast support was added
  - Refer to Chapter 3 "Using the Configuration Editors" and Chapter 4 "Using the Command Line Switches" for information about using this feature.
- Enhanced Trim Functionality was added
  - Refer to Chapter 3 "Using the Configuration Editors" and Chapter 4 "Using the Command Line Switches" for information about using this feature.
- T10 Data Protection (T10-PI) options were added
  - Refer to Chapter 3 "Using the Configuration Editors" and Chapter 4 "Using the Command Line Switches" for information about using this feature.

| **Version 7.1** |
|---|
| • Support of Microsoft Windows version 8.1 and version 10<br>  - Refer to the *Medusa Labs Test Tools Suite Version 7.1 Installation Guide*.<br>• Support for Linux 32- and 64-bit ARM processors<br>  - Refer to the *Medusa Labs Test Tools Suite Version 7.1 Installation Guide*.<br>• The addition of custom values for the **Set I/O Thread/CPU Affinity** setting in the "Planning Group Editor" on page 70 and the "Test Plan Editor" on page 73.<br>• **Enable Random Offset Alignment** check box has been added to enable the random offset alignment size for the Custom, Integrity, and Performance configuration editors.<br>  - Refer to Chapter 3 "Using the Configuration Editors" for information about using this feature.<br>• **Deduplication and Compression Testing** has been added to be used with the Compression & Dedup data pattern.<br>  - For information about using this feature with the graphical user interface (GUI), refer to the Pattern Tab for the Custom Configuration Editor on page 113. Refer to the Pattern Tab information for the Integrity, Socket, and the TCP App Simulation configuration editors as well.<br>  - Refer to "Deduplication/Compression Pattern (-l80)" on page 264 for information about using the `-l80` data pattern and its associated switch options for deduplication and compression testing.<br>• **Journal** and **Verifiy** has been added to record a log file of recent write operations so that in a simulated power loss, a log file is saved to preserve the status of the last several write operations. Then the log file can be examined using the verification process.<br>  - For information about using this feature with the graphical user interface (GUI), refer to "Steady State" on page 101 for Custom configurations or refer to "Steady State" on page 133 for Performance configurations.<br>  - Refer to "--journal Run I/O test with journaling enabled" on page 236 and "-V   Reverify Existing Data to a Specified Data Pattern/Verify Journaled Write Operations" on page 234 for information about using these features with the command line inputs. |

| **Version 7.0** |
|---|
| • **IOMeter Test Plan Import** allows IOMeter Configuration Files (.icf and .txt files) to be imported into MLTT as test plans. Refer to "Import Test Plans..." on page 54 for more information.<br>• **SSD Secure Erase** erases the data on a Solid State Drive (SSD) leaving it in a *clean* state.<br>  - Refer to "Format and Secure Erase Configuration Editor" on page 166 for information about using this feature with the graphical user interface (GUI). -Refer to "--secure-erase Erase the Target Device and Exit" on page 193 for information about using this feature with the command line inputs.<br>• **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.<br>  - Refer to "Trim Configuration Editor" on page 169 for information about using this feature with the graphical user interface (GUI).<br>  - Refer to "--trim Send Trim to Target" on page 195 for information about using this feature with the command line inputs.<br>• **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.<br>  - For information about using this feature with the graphical user interface (GUI), refer to "Steady State" on page 101 for Custom configurations or refer to "Steady State" on page 133 for Performance configurations.<br>  - Refer to "--steady-state Determine Steady State" on page 189 for information about using this feature with the command line inputs.<br>• **Latency Histogram** collects and displays latency histogram data per target using user-specified bins which are sorted by the magnitude.<br>  - For information about using this feature with the graphical user interface (GUI), refer to "General Tab" on page 100 for configuration information and refer to "Latency Histogram Tab" on page 93 for display information.<br>  - Refer to "--latency-histogram Collect Latency Histogram" on page 191 for information about using this feature with the command line inputs.<br>• **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.<br>  - Refer to "--smart S.M.A.R.T Monitoring" on page 221 for information about using this feature with the command line inputs. |

| **Version 7.7** |
|---|
| • Fixed support for Windows Systems that contain >64 logical CPUs<br>  - For more information, see "-T Set I/O Thread/CPU Affinity" on page 202<br>• Added NUMA awareness<br>  - For more information, see "-T Set I/O Thread/CPU Affinity" on page 202<br>• Added direct-access (DAX) to Linux PMEM devices (e.g. NVDIMM, CXL.mem)<br>  - Normal I/O generation using pain tool<br>• Added support for Linux ZNS devices<br>  - Normal I/O generation using pain and maim tools<br>  - Pass-Through mode ("- -ptio") with pain tool, including the option to use "Zone Append" command<br>  - Added new CLI option to reset ZNS zones. For more information, see "--nvme-reset-zone ZNS Zone Reset" on page 224 |

| **Version 7.7** |
|---|
| • Fixed support for Windows Systems that contain >64 logical CPUs<br>- For more information, see "-T Set I/O Thread/CPU Affinity" on page 202<br>• Added NUMA awareness<br>- For more information, see "-T Set I/O Thread/CPU Affinity" on page 202<br>• Added direct-access (DAX) to Linux PMEM devices (e.g. NVDIMM, CXL.mem)<br>- Normal I/O generation using pain tool<br>• Added support for Linux ZNS devices<br>- Normal I/O generation using pain and maim tools<br>- Pass-Through mode ("- -ptio") with pain tool, including the option to use "Zone Append" command<br>- Added new CLI option to reset ZNS zones. For more information, see "--nvme-reset-zone ZNS Zone Reset" on page 224 |

## 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] and Linux[2] (32-bit and 64-bit 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.

## How Medusa Labs Test Tools Work

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 4 shows a comparison of these tools.

**Table 4** Test Tools 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. Refer to Appendix N   "Sock Test Tool" for additional information.

## 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 113 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.

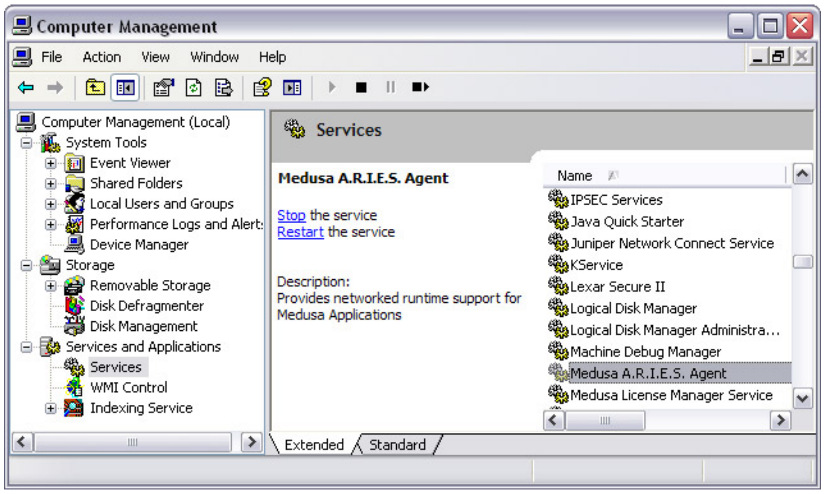
- Local license client
- Discovering other systems running MLTT in the network
- Mediate remote execution of MLTT

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



**Medusa A.R.I.E.S. Agent**

Stop the service
Restart the service

Description:
Provides networked runtime support for Medusa Applications

From the command line:

On Windows:

**`net stop maagent`** to stop the service
**`net start maagent`** to start the service

On Linux, use:

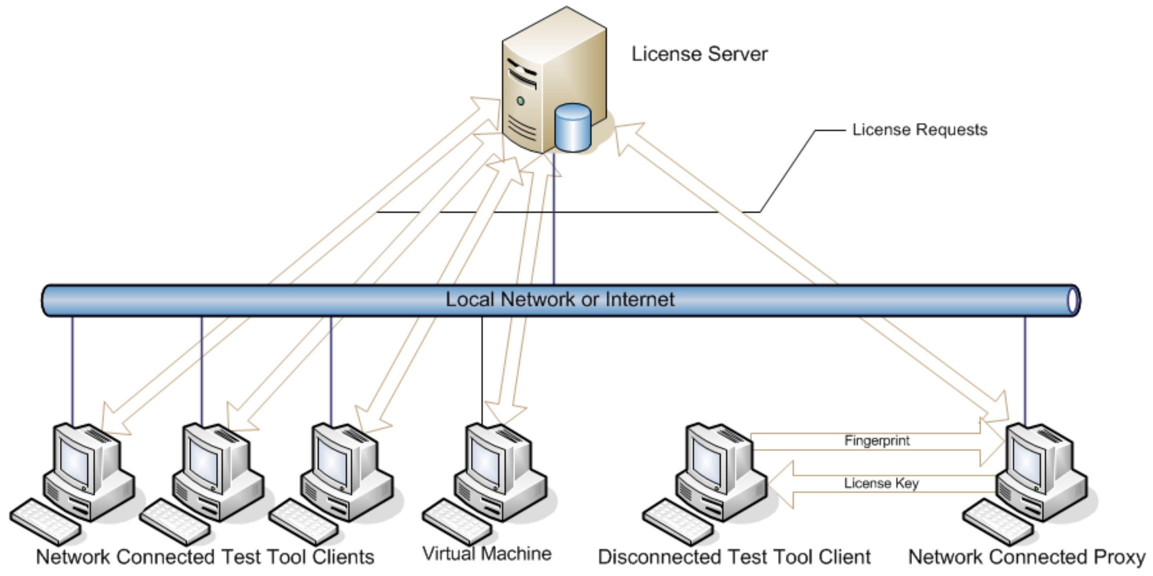
**`service maagent stop`** to stop the service
**`service maagent start`** to start the service

## 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.

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.

## Licensing Administration

This section includes information about commands supported by the Medusa License Management Administration (mlmadmin) tool.

- “Report Server State” on page 14
- “Report Client State” on page 14
- “Client User Configuration Property” on page 14
- “Client Auto Check-in Configuration Property” on page 15
- “Client Subnet Check-in” on page 15
- “Proxy Client Subnet Check-in” on page 16
- “License Server Pull” on page 16
- “License Server Net Sync” on page 16

## Report Server State

***Usage:***

```
mlmadmin -reportserverstate
```

***Description:***

Displays the current MLM server state.

## Report Client State

***Usage:***

```
mlmadmin -reportclientstate
```

***Description:***

Displays the current MLM client state.

## Client User Configuration Property

***Usage in the configuration file:***

```
MLM_USER=<user>
```

***Usage on the CLI:***

```
mlmadmin -lm-user <user>
```

License client **must be updated** to MLTT version 7.4+

***Description:***

To easily identify individual clients with checked-out licenses, the field `MLM_USER` can be set in the `MedusaTools.cfg` file. The property will be displayed in both the `mlmadmin -reportserverstate` and `mlmadmin -reportclientstate` output. The value specified can be a string of any length consisting of any character except for ";" since it signifies the end of the value. By default, the user property will be the logged-in user ID. This is not a required field and, as such, is initially absent from the configuration file. To set the property either add it manually by appending the field `MLM_USER=<user>` on a new line in the MedusaTools.cfg file or via command line using the `mlmadmin -lm-user <user>` command. The `MedusaTools.cfg` file is located in the `config` folder of where MLTT was installed. The default location of the `MedusaTools.cfg` file is:

Windows: `C:\Program Files\Medusa Labs\Test Tools\config`

Linux: `/opt/medusa_labs/test_tools/config`.

## Client Auto Check-in Configuration Property

#### *Usage in the configuration file:*

```
MLM_AUTO_CHECKIN=<1|0>
```

#### *Usage on the CLI:*

```
mlmadmin -lm-auto-checkin <0|1>
```

License client **must be updated** to MLTT version 7.4+

#### *Description:*

This property determines whether or now the running MLTT process - e.g. `pain`, `maim`, or `sock` - automatically attempts to check-in a license upon it's completion. A property value of MLM_AUTO_CHECKIN=1 indicates that the process will attempt to check-in the license prior to exit. If the license is currently in use - e.g. by another `pain`, `maim`, or `sock` process running on the same system - the license will not be checked in until the last running process prepares to exit. This is not a required field and is initially absent from the configuration file. The default value of this property is 0 which means that the process will NOT attempt to automatically check-in the license prior to exit. To set the `MLM_AUTO-CHECKIN` property either add it manually by appending the field `MLM_AUTO_CHECKIN` = followed by a "1" or "0" on a new line in the MedusaTools.cfg file or via command line using the command `mlmadmin -lm-auto-checkin 1` or `mlmadmin -lm-auto-checkin 0`. If the property value is set to 1 and the client has a license checked out, `mlmadmin -reportclientstate` and `mlmadmin -reportserverstate` will display the `AUTO_CHECKIN` property at the end of the "Client" attribute.The Medusa-Tools.cfg file is located in the `config` folder of where MLTT was installed. The default location of the `MedusaTools.cfg` file is:

Windows: `C:\ProgramFiles\Medusa Labs\Test Tools\config`

Linux: `/opt/medusa_labs/test-tools/config`

## Client Subnet Check-in

#### *Usage:*

```
mlmadmin -net -checkin
```

License client **must be updated** to MLTT version 7.4+

***Description:***

Directs all clients within the broadcast subnet of the host system that is executing the command to check-in the licenses if they are not in use - e.g. by another `pain`, `maim`, or `sock` process running on the same system.

## Proxy Client Subnet Check-in

***Usage:***

```
mlmadmin -client <proxy> -net -checkin
```

License client **must be updated** to MLTT version 7.4+

***Description:***

Directs a specific client, specified by their IP under the `<proxy>` argument, in a different broadcast subnet than the host system to perform the command `mlmadmin -net -checkin`. For more information please reference the "Client Subnet Check-in" section.

## License Server Pull

***Usage:***

```
mlmadmin -pull
```

License server **must be updated** to MLTT version 7.4+

***Description:***

Use `mlmadmin -pull` on the system that the license server is running on to have the license server request each registered client in the check-out list to check-in its license if not in use - e.g. by another `pain`, `maim`, or `sock` process running on the same system.

## License Server Net Sync

***Usage:***

```
mlmadmin -net -sync
```

License server and client must be updated to MLTT version 7.4+

***Description:***

Use `mlmadmin -net -sync` to direct license clients within the broadcast subnet to synchronize their license states, including those that have become `LOST`, with the server when the server's state becomes corrupt. A server's state can become corrupt when the license server's dongle is moved to a new server without going through the proper migration process. If you are having technical difficulties this can alternatively be fixed by an emergency dongle reset. For more information about emergency dongle resets please contact TechSupport-Medusa@viavisolutions.com.

## 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 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 - e.g. `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 - e.g. `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

**CAUTION**

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.

**CAUTION**

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#) x 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. Device drivers often 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 5** 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:**<br>• UAC is on<br>• Windows 8 Metro Live Tiles works correctly<br>• Catapult can see mapped network shares<br>• GUI can see mapped network shares | **Pros:**<br>• UAC is on<br>• Windows 8 Metro Live Tiles works correctly<br>• Catapult can determine local physical drive attributes<br>• GUI can determine local physical drive attributes<br>• Pain/Maim can run to local physical drives | **Pros:**<br>• Catapult can determine local physical drive attributes<br>• GUI can determine local physical drive attributes<br>• Pain/Maim can run to local physical drives<br>• Catapult can see mapped network shares<br>• GUI can see local network shares |
| **Cons:**<br>• Catapult cannot determine local physical drive attributes<br>• GUI cannot determine local physical drive attributes<br>• Pain/Maim cannot run to local physical drives | **Cons:**<br>• Catapult cannot see mapped network shares<br>• GUI cannot see mapped network shares | **Cons:**<br>• UAC is off<br>• Windows 8 Metro Live Tiles don't work |

**Table 5** 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 |
|---|---|---|
| **Work-arounds:**<br>• For GUI: right-click on GUI -> choose "Run as administrator..."<br>• For CLI: "Run as administrator..." a cmd.exe -> run the tools from within that console | | |
| **Notes:**<br>1 UAC OFF has been the normal (and one-and-only supported) mode of operation before MLTT 6.0.1.<br>2 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.<br>3 To disable the UAC in Windows 8 and above, 1) go to Local Security Policy setting "Local Policies/Security Options/User Account Control", 2) set "Run all administrators in Admin Approval Mode" to disable, and 3) reboot the PC. | | |

## MLTT Basics

This section includes the following information to provide basic understanding of MLTT when you are getting started:

### Prerequisites

Throughout this manual, the following terms are frequently used. The definitions are included for each term.

**Thread**: a unit of execution with its own context of CPU states.

**Process**: a container of one or more threads + resources shared among threads (address space, open file descriptors, synchronization objects, etc.)

**Blocking operation**: OS suspends the calling thread during the duration of the operation and schedules the thread to be runnable after the completion of the operation.

**Non-blocking operation**: OS does not suspend the calling thread for the duration of the operation.

**Synchronous I/O**: calling thread is blocked until the I/O is completed.

**Asynchronous I/O**: calling thread is not blocked when the I/O is queued with the OS and can continue to execute - “do other stuff” - while the I/O is pending. “Doing other stuff” while an I/O is pending can include starting more I/O operations.

**I/O queue depth**: number of I/Os that can be queued concurrently.

**Synchronous I/O**: maximum queue depth per I/O thread is always 1.

**Asynchronous I/O**: maximum queue depth per I/O thread can be greater than 1.

**NOTE**

There is a per-device maximum and there is also a higher-level OS limit. Per-device maximum for concurrent command queuing is some fixed size. OS-level maximum is typically dynamically determined by available system resources.

It is worth noting that “non-blocking I/O” and “asynchronous I/O” are not synonymous. By necessity, you need “non-blocking I/O” to implement “asynchronous I/O”. The I/O queuing part of an asynchronous I/O system should be a non-blocking operation.

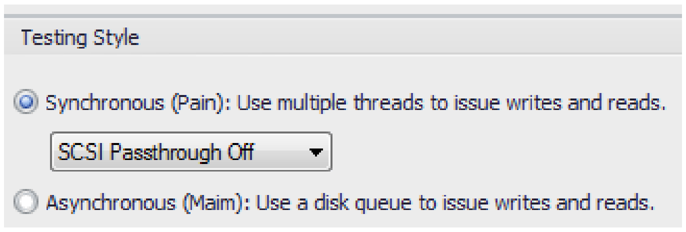
## Fundamental Concepts in MLTT

An MLTT I/O process: one running instance of “pain” or “maim”.

**CLI (Command Line Interface):**

– “`pain`” for synchronous I/O
– “`maim`” for asynchronous I/O

**GUI (Graphical User Interface):** “Testing Style”



Testing Style

(●) Synchronous (Pain): Use multiple threads to issue writes and reads.

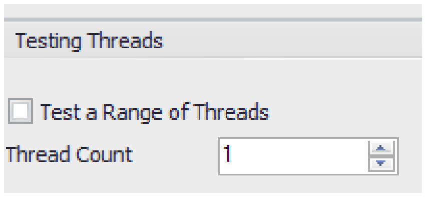
SCSI Passthrough Off

( ) Asynchronous (Maim): Use a disk queue to issue writes and reads.

I/O threads per process:

**CLI:** “`-t`”

**GUI:** “Thread Count”



Thread Count: 1

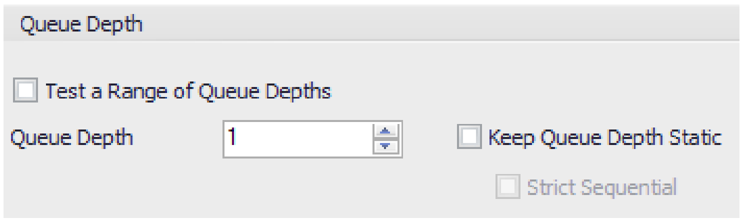
**NOTE**

“-t” and “Thread Count” are per-target. Therefore, the total number of I/O threads per process is “-t” or “Thread Count” number specified multiplied by the number of targets specified for that process.

I/O Queue depth per I/O thread

**CLI:** “`-Q`” (for “pain”, it is always “`-Q1`” and any other value is ignored)

**GUI:** “Queue Depth” (only available if “Asynchronous” is chosen)



Queue Depth: 1

**NOTE**

MLTT queues the I/O with the OS, not the device. Therefore, being able to specify a queue depth number that is greater than the device maximum does not imply that MLTT can magically queue more commands than what the device can handle.

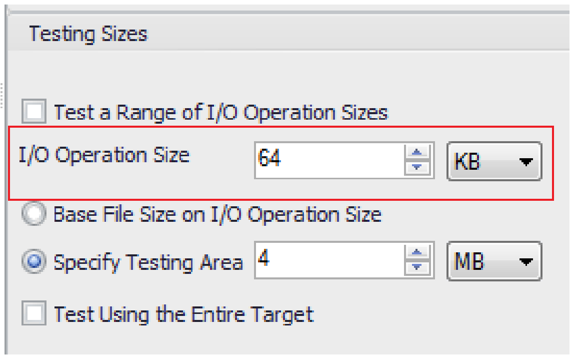
**NOTE**

“`-Q`” and “`Queue Depth`” are per-I/O thread. Therefore, the maximum number of I/Os that one running pain/maim process can queue is “Queue Depth” number multiplied by the “`Thread Count`” number multiplied by the number of targets specified for that process.

Buffer size per I/O operation

**CLI:** “`-b`”

**GUI:** “I/O Operation Size”


Testing Sizes
☐ Test a Range of I/O Operation Sizes
I/O Operation Size 64 KB
○ Base File Size on I/O Operation Size
● Specify Testing Area 4 MB
☐ Test Using the Entire Target

This is the amount of data MLTT will transfer to/from the OS in one read or write request to the OS.

**NOTE**

For “large” buffer sizes, the OS and/or the driver might break it down to multiple native commands of smaller transfer sizes. That detail is not visible to MLTT.

**Sequential-access I/O vs random-access I/O**: determines how each I/O thread determines the next I/O location (details in a later section.)

**“File Size” per thread**: determines the per-thread I/O operation area (details in a later section.)

**Read/write mix**: determines how each I/O thread determines if the next I/O operation should be a read or a write (details in a later section.)

## MLTT Storage Target Types

### Physical

This is a “disk” as presented by an HBA to the OS.

The actual topology below the HBA does not matter to MLTT.

- It could be a VLUN within an actual physical disk configured before the OS boots.
- It could be any number of hardware RAID configurations involving several actual physical disks.
- There may be multi-path configured.
- It could be a high-bandwidth device acting as a memory-only NUMA node.
- It may not even be a physical disk but a regular file on an NTFS file system on a remote system presented as a “disk” (HBA emulation, software iSCSI client, etc.)

A “disk” is one big flat “file” for MLTT to access.

### Logical

This is a “volume” created by the OS on top of one or more “disks”.

It may have RAID-like characteristics, but the constructs are above the HBA (i.e “it’s all software”.)

- Mirrored
- Spanning
- Play around in Windows Disk Management

As with a “disk”, a “volume” is one big flat “file” for MLTT to access.

### File System

A file system “target” for MLTT is just a regular file residing in a formatted file system.

- You can see the “target” file in Windows Explorer or with “`dir`” command.
- It can be a network file system, such as:
  - CIFS (aka “SMB”, “Windows File Sharing”)
  - NFS

A way to test NAS devices which are generally mounted as network shares.

**NOTE**

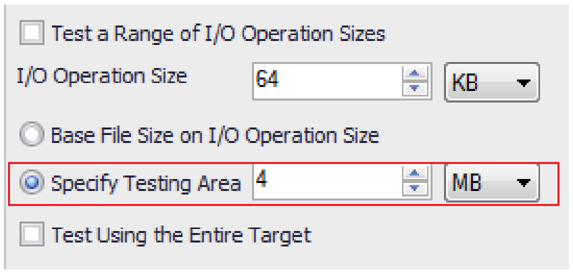
If the specified file system target is a directory, pain/maim will automatically append the path separator (‘/’ or ‘\’, depending on the OS) if necessary and append “targetfile.dat” as the file name.

## I/O Area Size

In MLTT, each thread is given an I/O area (“File Size”). The I/O areas of the I/O threads within each pain/maim process do not overlap unless a shared offset is specified using “-X”. An I/O thread never accesses areas outside its own (if it does, it is a serious bug).

**CLI:** Just the size, e.g, “`4MB`”, or “`--file-size=4MB`”.

**GUI:** "Specify Testing Area"



[ ] Test a Range of I/O Operation Sizes
I/O Operation Size: 64 KB
○ Base File Size on I/O Operation Size
● Specify Testing Area: 4 MB
[ ] Test Using the Entire Target

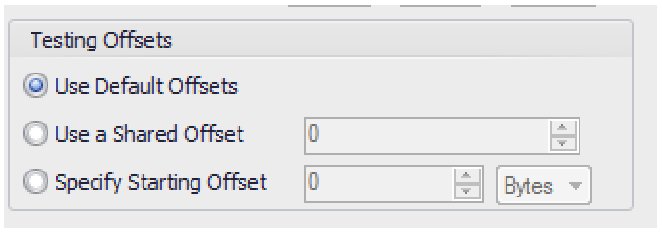
The threads' I/O areas are sequentially adjacent to each other – unless "shared offset" is specified (see below). Also, you can specify the starting offset for the target. The I/O area of the first I/O thread to the target starts at this offset.

For "Physical" disks and "Logical" volumes, there is a default starting offset of "1MB". For file system targets, the default offset is "0".

**CLI:**

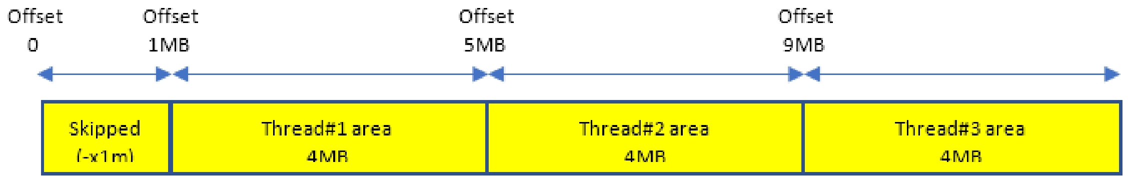
- "`-O`" override the default offset
- "`-x`" set a specific starting offset ("`-x0`" is same as "`-O`")
- "`-X`" set a specific shared offset that is shared by all I/O threads in the process. When the shared offset is specified, then all I/O threads share the same I/O area.

**GUI:** "Testing Offsets"



Testing Offsets
● Use Default Offsets
○ Use a Shared Offset: 0
○ Specify Starting Offset: 0 Bytes

**Figure 3** Example: I/O Areas using "`-x1m -t3 4MB`"



**NOTE**

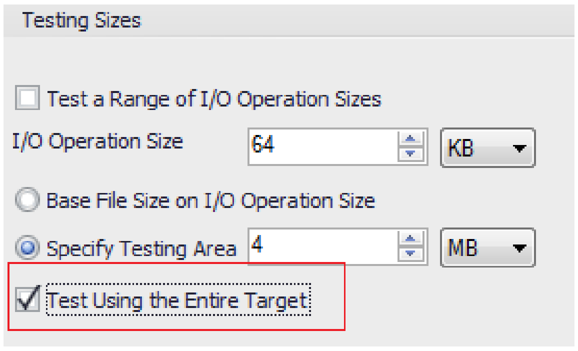
In non-full device mode, MLTT enforces "file_size = N* buffer_size * queue_depth" for some integer "N > 0". If the user specified "File Size" does not meet this restriction, then it is adjusted. For example, the file size must be an exact multiple of "buffer_size times queue_depth".

When the target is “Physical” or “Logical”, you can specify to use the full device size to calculate the actual per-thread I/O area.

**CLI:**

- Macro “`-m18`” for full-device sequential-access I/O.
- Macro “`-m17`” for full-device random-access I/O.
- Or, specify “`--full-device`” regardless of access mode.

**GUI:** check “Test Using the Entire Target”


Testing Sizes

Test a Range of I/O Operation Sizes: unchecked
I/O Operation Size: 64 KB
Base File Size on I/O Operation Size: not selected
Specify Testing Area: selected, 4 MB
Test Using the Entire Target: checked

**NOTE**

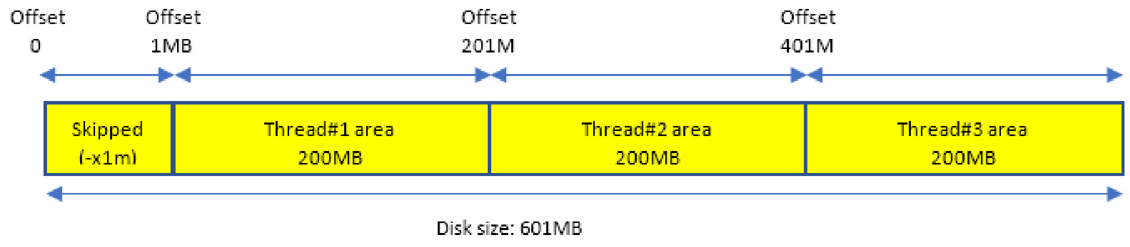
Full device specifications are ignored on File System targets.

When full device testing is specified, the actual per-thread test area is calculated based on the actual disk or volume size:

“(disk_size – starting_offset) divided by thread_count”

When both “`--full-device`” and sequential-access mode are specified, the file size parameter becomes an internal “stroke size”. Each thread will cover its I/O area in the increments of this “stroke size”.

**Figure 4** Example: I/O Areas using “`-x1m -t3 4MB --full-device`” running to a 601MB disk as an example



Each thread’s I/O area is now a value calculated using the actual disk size of 601MB rather than the “4MB” file size value that is specified by the user. For sequential I/O, each thread will cover its 200MB area in 4MB “strokes” (the user -specified “File Size”). For instance:

- Write 4MBs
- Rewind 4MBs
- Read 4MBs
- Write next 4MBs
- Rewind 4MBs
- Read 4MBs,
- etc.

In this way, each thread covers its 200MB area in “200MB / 4MB = 50 strokes”.

Most often, the per-thread I/O area calculation numbers do not work out nicely for full-device access given some user-specified values. When this occurs, some fudge factors are involved to provide a “best fit”. In general, MLTT will try to honor

- user-specified thread-count,
- user-specified queue depth,
- user-specified buffer size, and
- make sure every byte between starting offset and the end of the target is included in the I/O area.

The user-specified “File Size” (“stroke size”) can be drastically adjusted.

**NOTE**

In MLTT, there’s no such thing as “full-device file system” target. Any option that specifies or implies “full device” is ignored for “file system” targets.

## Sequential-Access I/O (General Description)

In sequential-access mode, the I/O thread issues the next I/O to the directly adjacent area in the target. The next I/O offset is last I/O offset plus “buffer size” for forward sequential or minus “buffer size” for backward sequential.

**Figure 5** Forward sequential example using “-b64k”

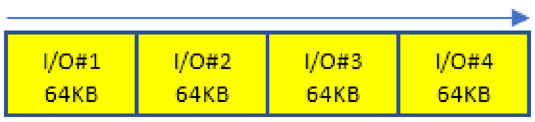
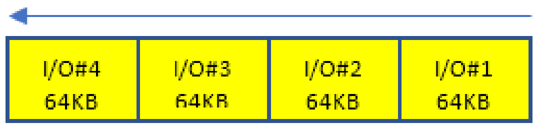


**Figure 6** Backward sequential example using "`-b64k`"

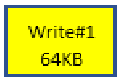


In sequential-access mode, once the end of the thread's I/O area is reached, this marks the end of 1 sequential iteration, and the "FOP" ("File Operations") counter is incremented. I.e. 1 sequential iteration over the designated I/O area for the thread means 1 completed "FOP" for that thread. After reaching the end of the I/O area, the sequential-access mode test "rewinds" to the starting offset of the I/O area and performs the next sequential I/O pass over the I/O area.

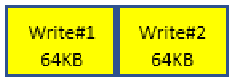
In the default sequential-access mode, I/O occurs in pairs of "write pass" and "rewind-and-read pass" over the "file size" area per thread. By default, during the read-pass, the data comparison is also performed to check for data corruptions. In this default configuration, it is the completion of each pair of the "write pass" followed by "rewind-and-read pass" that completes 1 FOP (NOT the completion of just the "write pass" or the "read pass").

For example: "`pain -b64k 256k`" (buffer size 64KB, file size 256KB):

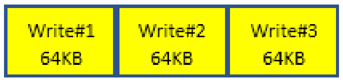
- Write to first 64KB of the I/O area (64KB of 256KB written):



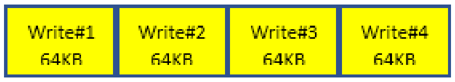
- Write to next 64KB of the I/O area (128KB of 256KB written):



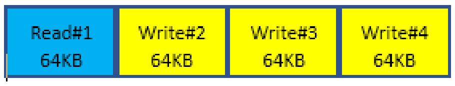
- Write to next 64KB of the I/O area (192KB of 256KB written):



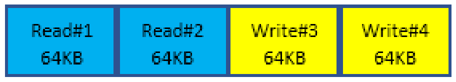
- Write to the last 64KB of the I/O area (256KB of 256KB written):



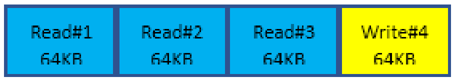
- Rewind. Read from the first 64KB of the I/O area – compare data (64KB of 256KB read):



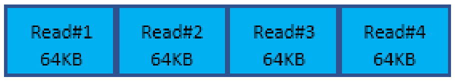
- Read from next 64KB of the I/O area – compare data (128KB of 256KB read)



- Read from next 64KB of the I/O area – compare data (192KB of 256KB read)



- Read from final 64KB of the I/O area – compare data (256KB of 256KB read)



- 1 FOP complete for thread
- Rewind – issue Write #5, 6, 7, 8
- Rewind – issue Read #5, 6, 7, 8 – compare data after each read
- 2 FOPs complete for the thread
- etc.

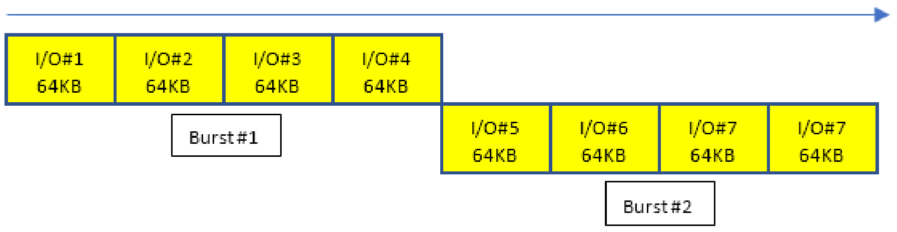
## Asynchronous Sequential-Access I/O, Burst Queueing

For asynchronous I/O, sequential-access mode is further characterized as “burst queueing” or “continuous queueing”. The difference between the two types is relevant only if “queue depth” is greater than 1.

In burst queueing,

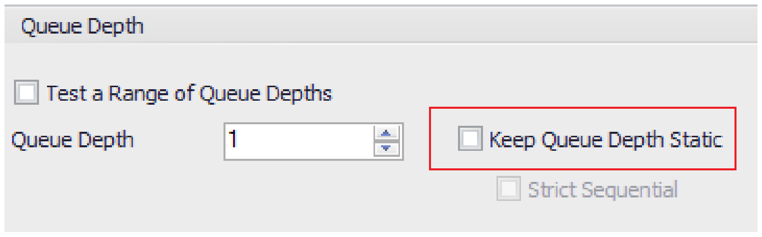
- Each I/O thread queues “queue depth” number of I/Os to the OS.
- Each I/O thread then waits for, and handles, all of its pending I/Os to complete.
- When all of its pending I/Os are completed, each I/O thread issues the next “queue depth” number of I/Os.
- etc.

Example using “`-b64k -Q4`”



**CLI:** Burst queueing is specified using “`-m11`” (default for “`maim`”) or “`-m18`”.

**GUI:** Burst queueing is specified by unchecking “Keep Queue Depth Static”.



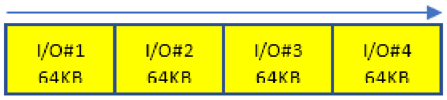
Queue Depth: 1
☐ Keep Queue Depth Static

## Asynchronous Sequential-Access I/O, Continuous Queueing

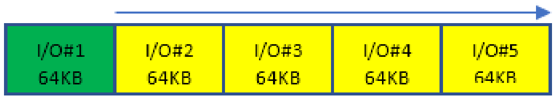
In continuous queueing mode, after the initial burst queueing of "queue depth" number of I/Os, the I/O thread immediately issues the next I/O as soon as one of the pending I/Os completes.

Example using "`-b64k -Q4`":

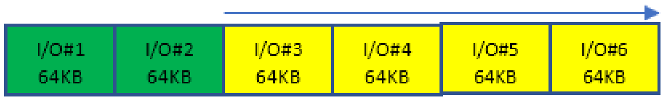
- Initial burst of 4 sequential I/Os, 64KBs each.



- As soon as I/O#1 completes, queue I/O#5 at next sequential offset.

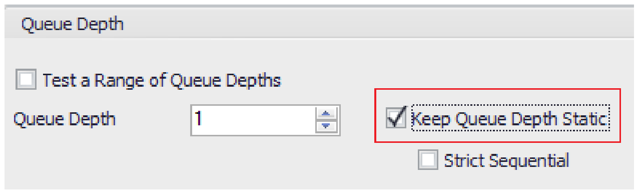


- As soon as I/O#2 completes, queue I/O#6 at the next sequential offset.



**CLI:** Continuous queueing is specified with "`-m1`" or "`-m16`".

**GUI:** "`-m16`" is specified by selecting "Keep Queue Depth Static"


Queue Depth

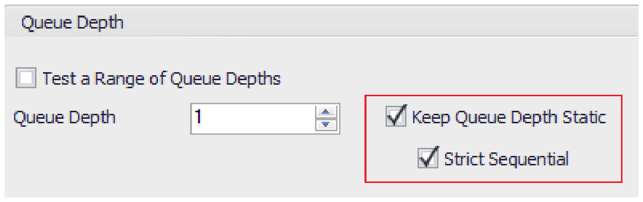
☐ Test a Range of Queue Depths

Queue Depth: 1

☑ Keep Queue Depth Static

☐ Strict Sequential

**GUI:** "`-m1`" is specified by selecting both "Keep Queue Depth Static" and "Strict Sequential".


Queue Depth

☐ Test a Range of Queue Depths

Queue Depth: 1

☑ Keep Queue Depth Static

☑ Strict Sequential

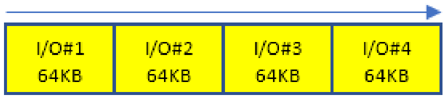
The difference between "`-m1`" ("Strict Sequential") and "`-m16`" (NOT "Strict Sequential") is in how MLTT handles I/O completion notifications that arrive "out-of-order".

In "`-m16`", the algorithm to calculate the next I/O offset is:

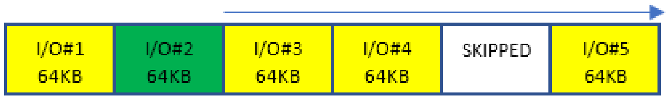
"offset_of_just_completed_I/O plus buffer_size multiplied by queue_depth"

Given the above I/O sequence pictures, after the initial burst of 4 I/Os are queued, consider what would happen if I/O#2 completes before I/O#1. According to the “-m16” offset calculation, the IO#5 will skip a 64KB area that would be accessed if I/O#1 would have completed.

Initial burst of 4 sequential I/Os, 64KBs each, same as before.



However, this time, I/O#2 completes first. Queue I/O#5 at I/O#2’s offset + 64KB x 4.



The “SKIPPED” area will be accessed as soon as I/O#1 completion notification arrives.

This “-m16” algorithm was devised without knowing that asynchronous I/O completion notifications can arrive out-of-order.

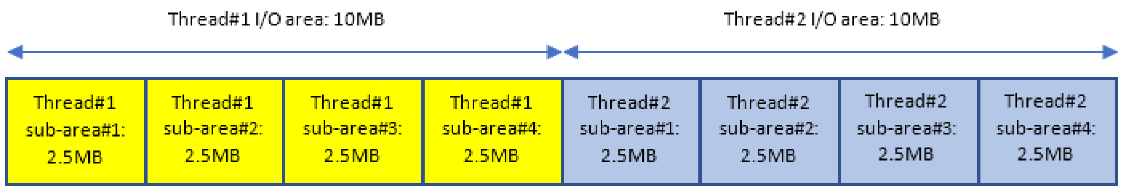
However, “`-m1`” accounts for out-of-order completions and makes sure the next I/O doesn’t skip the next adjacent area (thus “Strict Sequential”).

## Random-Access I/O (General Description)

For random-access I/O, each I/O thread keeps up to “queue depth” number of concurrent I/Os to randomly chosen offsets within its I/O area. Furthermore,

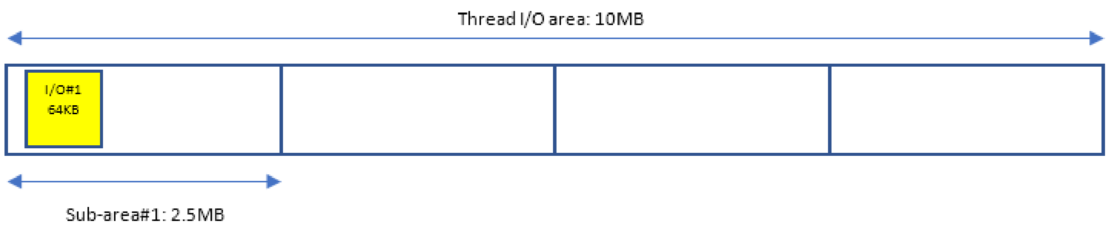
- each thread’s I/O area is divided into “queue depth” number of adjacent sub-areas, and
- each thread issues only 1 I/O at a time to each sub-area.

Consider the asynchronous I/O scenario, where “queue depth” per I/O thread can be greater than 1, as a generalized case. Using “`-b64k -Q4 -t2 10MB`” as an example, each thread’s 10MB I/O area is divided into “10MB file size divided by queue depth 4”, or 2.5MB, sub-areas.

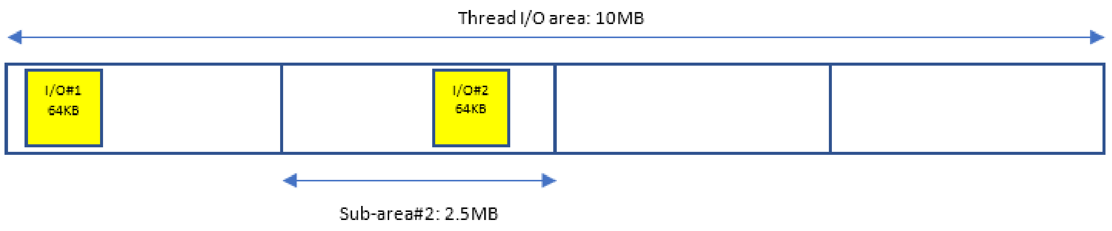

Thread#1 I/O area: 10MB
Thread#2 I/O area: 10MB
Thread#1 sub-area#1: 2.5MB
Thread#1 sub-area#2: 2.5MB
Thread#1 sub-area#3: 2.5MB
Thread#1 sub-area#4: 2.5MB
Thread#2 sub-area#1: 2.5MB
Thread#2 sub-area#2: 2.5MB
Thread#2 sub-area#3: 2.5MB
Thread#2 sub-area#4: 2.5MB

When I/O starts, each thread first queues the initial burst of 4 I/Os, 1 per each I/O sub-area:

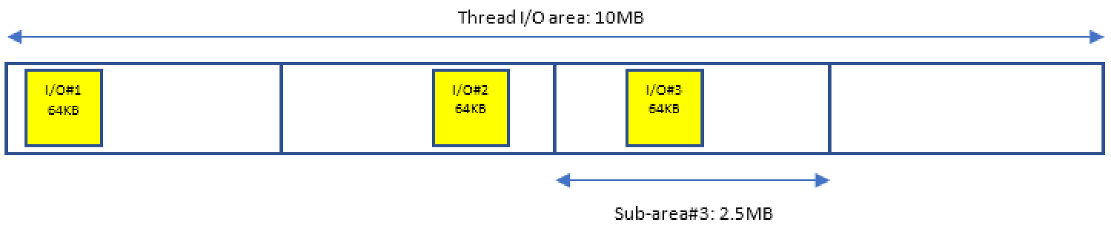
- Determine a random offset within sub-area#1 -> queue a 64KB I/O#1 to that offset



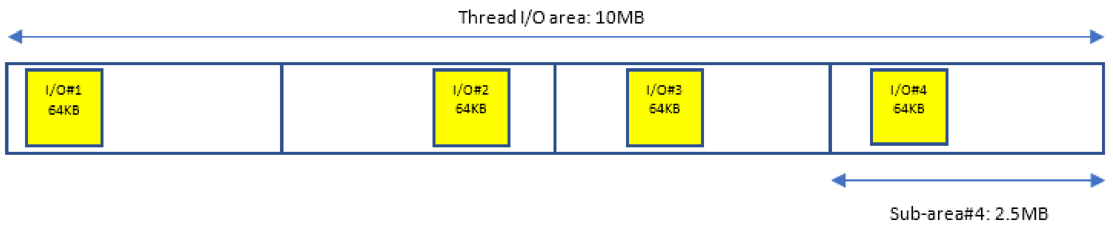
- Determine a random offset within sub-area#2 -> queue a 64KB I/O#2 to that offset



- Determine a random offset within sub-area#3 -> queue a 64KB I/O#3 to that offset

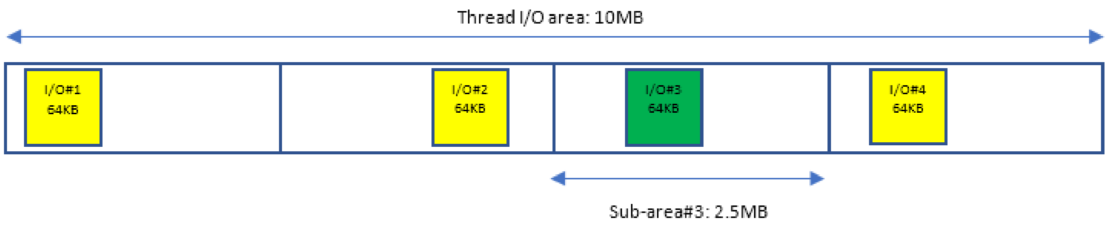


- Determine a random offset within sub-area#4 -> queue a 64KB I/O#4 to that offset

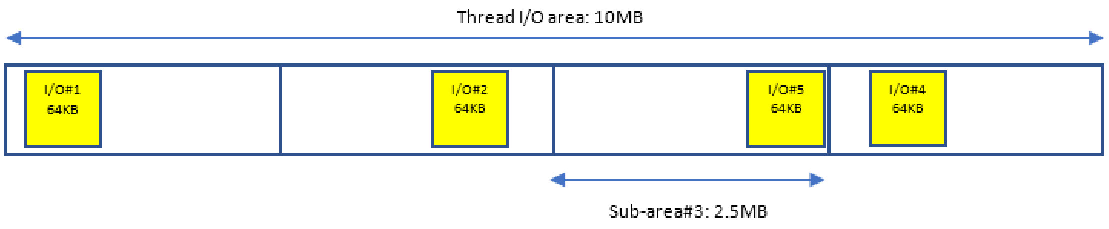


Once a thread queues the initial burst of four I/Os ("queue depth" number) within its I/O area, 1 per randomly chosen offset within a sub-area, it then waits for any one of its four pending I/Os to complete. As soon as an I/O of a sub-area completes, the thread handles that I/O completion and immediately issues another I/O to a new randomly determined offset within that same sub-area. For example, continuing with the example, assume I/O#3 completes first. Note that out-of-order completions occur.

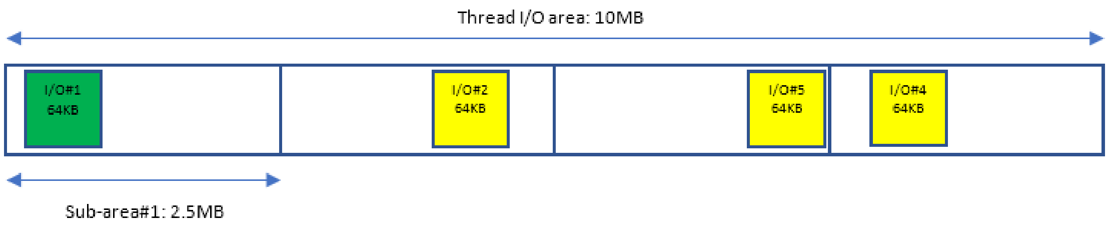
- I/O#3 in sub-area#3 completed



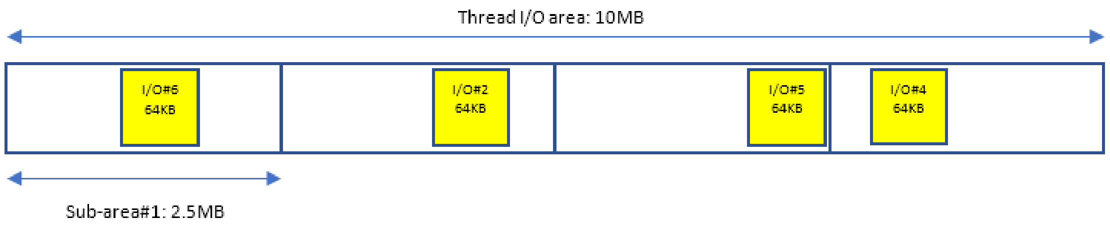
- Determine a random offset within sub-area#3 -> queue a 64KB I/O#5 to that offset



- I/O#1 in sub-area#1 completed



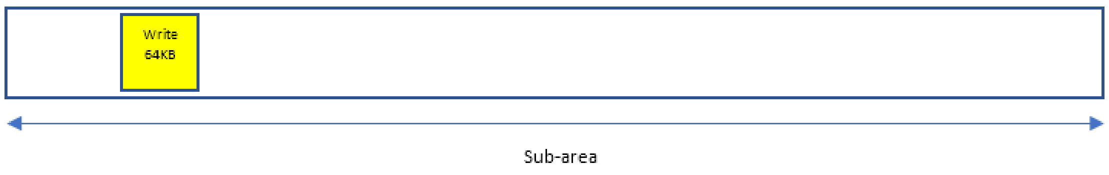
- Determine a random offset within sub-area#1 -> queue a 64KB I/O#6 to that offset



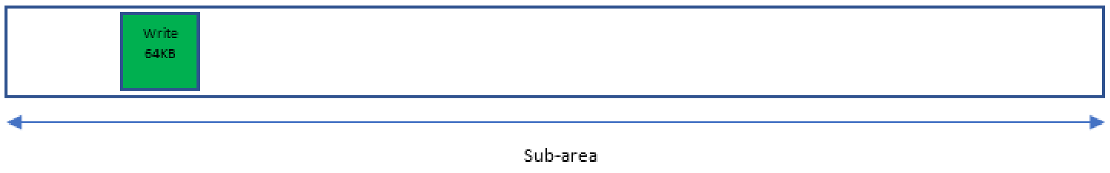
- Etc.

In default random-access mode, a write operation is immediately followed by a read operation from the same location followed by data comparison. As an example, with "`-b64k`", for each sub-area,

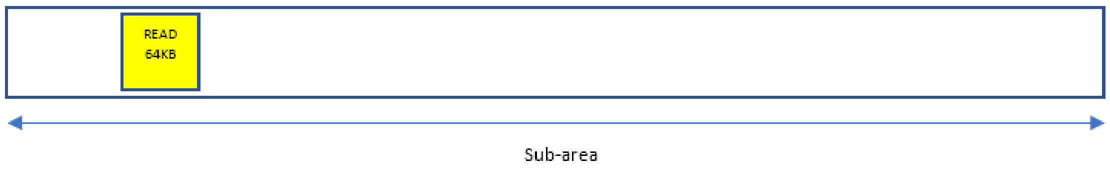
- Determine a random offset within sub-area -> queue a write operation to that offset



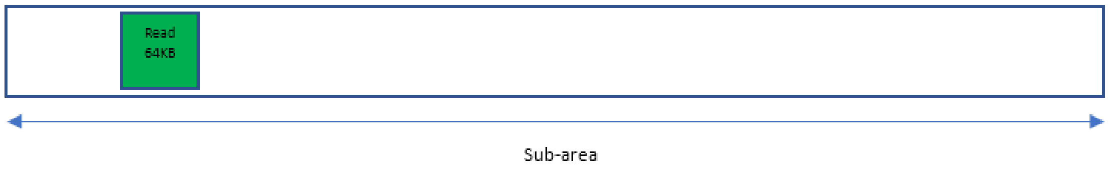
- Write operation completes



- Queue a read operation to read from the same offset



- Read operation complete



- The data read from the device should be identical to the data that was written to it. You can compare the two to confirm they are identical.

## Some Additional Considerations

**NOTE**

Unlike in sequential-access I/O, there's no "burst queueing" vs "continuous queueing" distinction in random-access I/O. In random-access mode, after the initial burst of "queue depth" number of I/Os, subsequent I/Os are always queued continuously (i.e. immediately upon successful completion of any of the pending I/Os).

Remembering that in synchronous I/O ("`pain`") the queue depth is always 1, you can specialize the above example for "queue depth" of 1 to consider the random-access synchronous I/O tests.

While each thread keeps up to "queue depth" number of I/Os queued within its I/O area, only 1 I/O is queued to a sub-area. By keeping concurrent I/Os in non-overlapping sub-areas, MLTT ensures that random-access I/Os do not overwrite each other. This in turn ensures that we can perform data corruption check during each I/O completion handling without worrying about "false data corruption" - data comparison failing due to MLTT I/Os themselves overwriting each other's data.

**NOTE**

A "false data corruption" is the worst kind of MLTT bug you can encounter – even more serious than a program crash.

The above example is for non-full device coverage case with a user-specified file size as the per-thread I/O area.

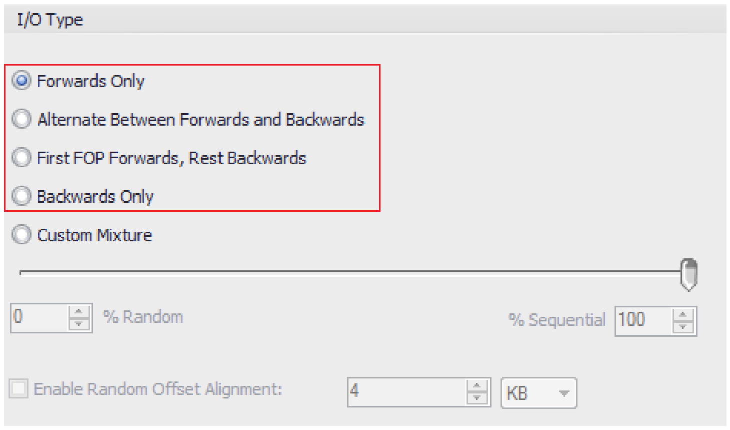
## Sequential-Access vs. Random-Access I/O

As covered in "Sequential-Access I/O (General Description)" on page 29, the basic concept of sequential-access I/O is that the next I/O is issued to an offset adjacent to the last I/O. Furthermore, in sequential-access mode, there is no direction change during the coverage of the I/O area until after the last I/O to the last I/O offset within the I/O area during an iteration.

In CLI, the sequential I/O can be specified with macro modes "`-m1`", "`-m11`", "`-m16`", "`-m18`" and the following switches:

- "`-B0`" (default): forward sequential
  - I/Os are issued sequentially from the lowest offset to the highest offset within the I/O area.
  - After the last I/O to the highest offset, next sequential iteration starts at the lowest offset of the I/O area (i.e. "rewind")
- "`-B1`": alternate forward-backward per "FOP" (recall what a "FOP" is from section 6).
  - I/Os are issued sequentially from the lowest offset to the highest offset within the I/O area.
  - After the last I/O to the highest offset, the next sequential iteration starts at the highest offset. I/Os are issued from the highest offset to the lowest offset within the I/O area.
  - After the last I/O to the lowest offset, the next sequential iteration starts at the lowest offset of the I/O area.
  - Repeat
- "`-B2`": first FOP forward, rest backward
- "`-B3`": backward sequential

GUI offers direct parallels for “-B”, but which “-m” is chosen depends on burst vs. continuous and full-device or not.



(● Forwards Only)
○ Alternate Between Forwards and Backwards
○ First FOP Forwards, Rest Backwards
○ Backwards Only
○ Custom Mixture

0 % Random
% Sequential 100

☐ Enable Random Offset Alignment: 4 KB

Random access is specified in CLI with either “`-m17`” macro or with a more elaborate “`-%x,f,b`” mix.

- “`-m17`”: 100% random-access AND full-device access
- “`-%x`”: a random offset value between the lowest offset and the highest offset within the I/O sub-area
- “`-%f`”: next offset will be “forward sequential” from the last offset
- “`-%b`”: next offset will be “backward sequential” from the last offset

**NOTE**

“`-m17`” is same as “`-%x100 --full-device`”

“`-%f100`” is the same as “`-B0`”

“`-%b100`” is the same as “`-B3`”

There is some confusion regarding “`-%f`” and “`-%b`”. Consider “`-B1`”, which specifies alternating forward/backward sequential passes. This is, obviously, “50% forward, 50% backward” – i.e. 50% of all I/Os are forward adjacent to the previous I/O, and the other 50% of all I/Os are backward adjacent to the previous I/O. However, “`-B1`” is NOT “`-%f50 -%b50`”, which also specifies “50% forward, 50% backward”. In MLTT, “`-B1`” performs sequential-access I/O while “`-%f50 -%b50`” (or the shorthand “`-%f50,b50`”) performs random-access I/O.

Recall the statement from the beginning of this section: “in sequential-access mode, there is no direction change during the coverage of the I/O area until after the last I/O to the last I/O offset within the I/O area during an iteration”.

With “`-B1`”, all I/Os occur in one direction from one end of I/O area to the other. The direction change occurs only at the end of one sequential iteration over the I/O area. However, with “`-%f50,b50`”, the direction change is considered for every I/O. I.e, for each I/O:

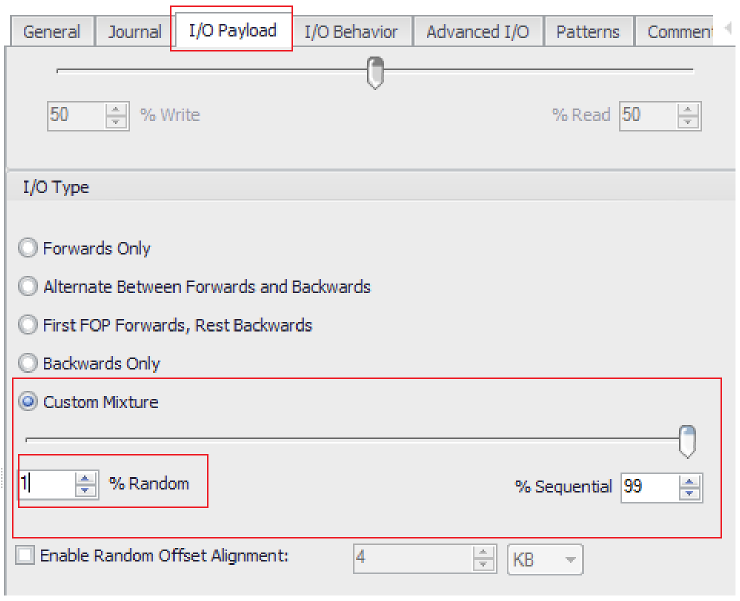
- Flip a coin.
- Heads -> next I/O is forward-sequential from the previous I/O.
- Tails -> next I/O is backward-sequential from the previous I/O.
- I.e. there's no concept of going from one end and progressing until the other end in one direction. However,
  - There is a non-zero probability of getting only "heads" or only "tails" for every coin flip.

Therefore, if a custom access profile is specified,

- `-%f100` is sequential-access.
- `-%b100` is sequential-access.
- `-%x100` is random-access.
- Any combination of the following with non-zero weight value (probability) is random access:
  - "`-%x`" and "`-%f`"
  - "`-%x`" and "`-%b`"
  - "`-%x`" and "`-%f`" and "`-%b`"
  - "`-%f`" and "`-%b`"

In the GUI, random-access I/O is specified in following ways.

- Under "I/O Payload" -> "I/O Type", "Custom Mixture" with non-zero value for "% Random".



50 % Write % Read 50

I/O Type

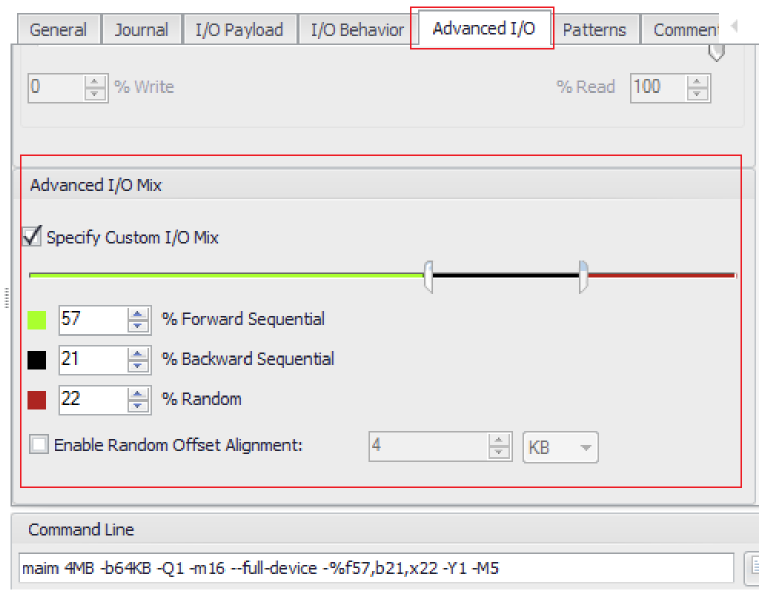
- Forwards Only
- Alternate Between Forwards and Backwards
- First FOP Forwards, Rest Backwards
- Backwards Only
- [✓ Custom Mixture]

1 % Random % Sequential 99

Enable Random Offset Alignment: 4 KB

This is an extreme example where most I/Os will progress sequentially forward from the previous I/O, but, every once a while, an I/O will jump to a random location with the I/O area. Nevertheless, this is random-access because it is not 100% sequential.

- Access the full I/O profile parameters under “Advanced I/O”.



0 % Write     % Read 100

Advanced I/O Mix

✓ Specify Custom I/O Mix

🟩 57 % Forward Sequential
■ 21 % Backward Sequential
🟥 22 % Random

☐ Enable Random Offset Alignment: 4 KB

Command Line

```
maim 4MB -b64KB -Q1 -m16 --full-device -%f57,b21,x22 -Y1 -M5
```

## Reads, Writes, And Data Integrity Checking

There are different ways to specify how much of the operations is for read (data transfers from the target device) and how much is for write (data transfers to the target device). This also has implications on whether or not data integrity checking is possible.

The default sequential-access mode of “write pass, rewind-and-read pass with data comparison” was described in “Sequential-Access I/O (General Description)” on page 29. Likewise, “Random-Access I/O (General Description)” on page 34 covered write-read-compare for random-access I/O. In both cases, there is equal amounts of write and read operations (“50% read, 50% write”).

Adding the simple “-w” option makes this “100% write” (“write-only”). This is same for both sequential-access and random-access modes. Data comparison is not possible for “write-only” tests.

The “-r” option is for “read-only” tests. For sequential-access, it also has a “initial write-once” variant with or without data comparison. There is no initial write pass in any of the “read-only” specifiers for random access mode.

- “`-r`”
  - Sequential-access:
    - With “`-n`” also specified: “no initial write pass” and “no data comparison (-n)”; “100% reads”
    - Without “`-n`”: “initial write pass” and “with data comparison” – “read-only” after the initial write pass
  - Random-access: no-write pass, 100% reads, no data comparison
- “`-ro`”: also turns on “`-n`” and “`-o`”
  - Sequential-access:
    - Always performs initial write-pass once
    - No data comparison
  - Random-access: no-write pass, 100% reads, no data comparison

In addition, there is a “re-verify read-only” mode: “`-V`”. The purpose of “`-V`” is to perform a read-only test with data verification on a target which contains data written to it in a previous MLTT test. The re-verify test is applicable only to sequential-access. A typical work-flow:

1 Run 1 sequential iteration (“1 FOP”) of write-only test to a target.

2 Take the target offline or reboot the system.

3 Run “`-V`” to the same target, taking care to use exactly the same parameters as before (i.e same buffer size (“`-b`”), file size, data pattern “`-l`”, etc.) to ensure the data was correctly persisted.

This test covers the back-up/restore type of use-cases where you would archive the back-up data and store the backup disk somewhere off-line until data restoration from it is needed.

The newer “I/O profile” option (“`-%`”) offers a more flexible way to mix different amounts of reads and writes as well as the ability to associate different transfer lengths to the operations.

For example, “`-%r50,w50`” specifies “50% reads, 50% writes”. Recall that default I/O modes for sequential-access and random-access modes also perform equal number of reads and writes – i.e. also “50% reads, 50% writes”. However, for default modes, the write and read sequences are very deterministic and allows data comparison. The 50/50 breakdown specified by the I/O profile mode, “`-%r50,w50`”, however, is very different, because the sequence is randomly generated. I.e.

- Determine the next I/O location (review sequential- and random-access)
- Flip a coin.
- Heads -> next I/O is a read.
- Tails -> next I/O is a write.

This is a test where the primary purpose is to see the performance impact of mixing a varying ratio of reads vs writes. MLTT currently cannot do any data comparison in this mode. Consider, for example, an unbalanced mix: "`-%r30,%w70`".

The MLTT data comparison design is such that it reads and verifies what it wrote – i.e. the comparison would have to be done by reading back the 70% of operations which are writes. This also means that it changes the actual ratio of reads/writes to something very different from what the user specified. You cannot have only 30% of the operations being reads when all the writes – specified to be 70% of all operations - need to be read-back for verification. A different scheme can be designed and implemented to allow some data comparison in the I/O profile read/write mix mode. As it is in the present state, there is no data comparison in custom read/write mix mode by design.

**NOTE**

Journal verification mode ("`--journal`") does allow some data comparison and verification of data written by custom read/write mix test.

One other capability of the I/O profile read/write mix specification is to be able to attach different transfer sizes at different probabilities. E.g.

```
-%r20@4k,r30@16k,w20@16k-32k,w30 -b128k
```

This set of specifications says

- `-%r20@4k`: 20% of all operations will be 4KB reads
- `-%r30@16k`: 30% of all operations will be 16KB reads
- `-%w20@16k-32k`: 20% of all operations will be writes of random transfer sizes between 16KB to 32KB
- `-%w30`: 30% of all operations will be 128KB writes.

**NOTE**

There's no "`@128k`" attached to this spec – when there is no transfer size attached to a spec, the global "-b" value is attached to it (see "-b128k" above).

The "-%r" and "-%w" specifications are accumulative. With most MLTT CLI options, when you repeat a same option more than once, the most recent one overrides the previous ones. E.g.

```
pain -b128k -b1m
```

In this example, "-b" option is specified twice. The first "`-b128k`" is ignored, so the command becomes "`pain -b1m`".

With "-%r" and "-%w", all occurrences are used together to form the final read/write mix specifications. The example above could have been specified with separate arguments:

```
-%r20@4k -%r30@16k -%w20@16k-32k -%w30 -b128k
```

Pain/maim allows an implicit (100 – x) notation if (and only if) exactly 1 “-%r” or “-%w” specification is given in the command line AND if no explicit transfer size is attached to that spec. E.g.

```
pain -%r25
```

is the same as

`pain -%r25 -%w75` (or the short-hand pain `-%r25,w75`)

I.e. “25% reads, 75% writes).

However, if you attach a transfer size to it, the implicit (100 – x) notation does not apply.

```
pain -%r25@4k
```

The above example specifies “100% of all operations are 4KB reads”.

Next point of clarification is why “25%” becomes “100%” in the last example? That is because the probability weight values are not percentages (although, you can use percentage values if that makes it easy for you). The final scale and ratio is determined by the sum of all weight values specified. E.g.

```
pain -%r2,w3
```

The combined weight value is “2 + 3 = 5”. Therefore, the amount of reads specified is “2/5 x 100 = 40%”, and the amount of writes specified is “3/5 x 100 = 60%”. “`-%r40,%w60`” would have yielded the same result if you feel more comfortable thinking in terms of percentages. If you think in terms of “2 to 3 (2:3) read-write ratio”, then “-`%r2,w3`” is a direct translation of that notation.

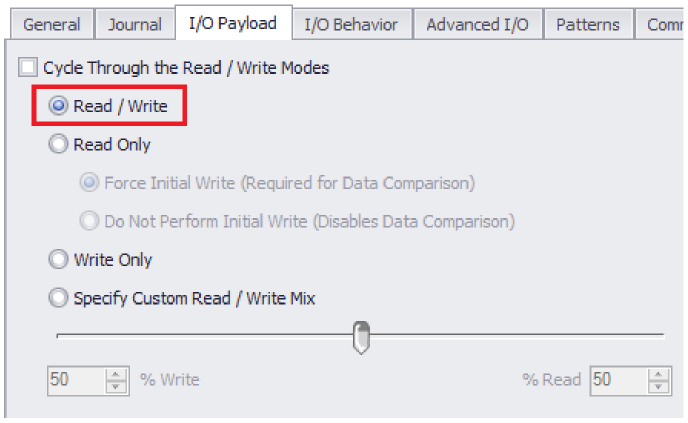
Therefore, “`pain -%r25@4k`” means “25/25 x 100 = 100%” reads at 4KB transfer size.

**NOTE**

This ratio evaluation rule also applies to the I/O location specifiers “`-%f/b/x`”.
Note that the weights are not percentages but can be specified in terms of percentages.

The GUI equivalents of the aforementioned I/O operation specifiers are as follows:

- "Normal" read/write mode, data comparison possible (i.e. no "`-r`", "`-w`", "`-%r/w`" specified).



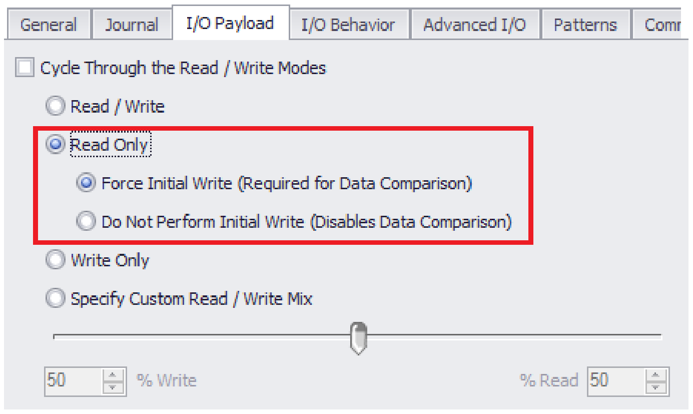
General | Journal | I/O Payload | I/O Behavior | Advanced I/O | Patterns | Comr

☐ Cycle Through the Read / Write Modes

[✓ Read / Write]
Read Only
[✓ Force Initial Write (Required for Data Comparison)]
Do Not Perform Initial Write (Disables Data Comparison)
Write Only
Specify Custom Read / Write Mix

50 % Write    % Read 50

- "`-r`", initial write pass, data comparison possible.



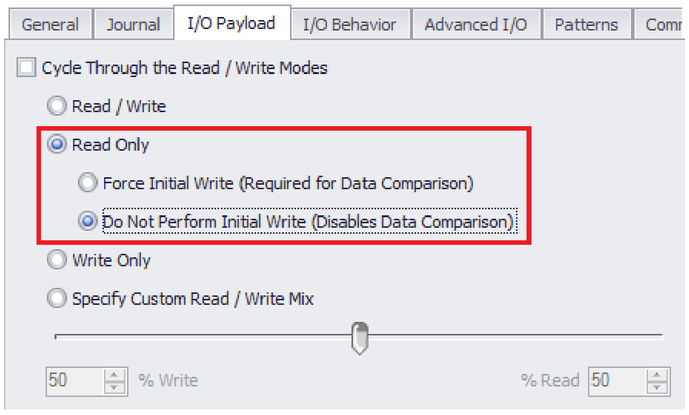
General | Journal | I/O Payload | I/O Behavior | Advanced I/O | Patterns | Comr

☐ Cycle Through the Read / Write Modes

Read / Write
[✓ Read Only]
[✓ Force Initial Write (Required for Data Comparison)]
Do Not Perform Initial Write (Disables Data Comparison)
Write Only
Specify Custom Read / Write Mix

50 % Write    % Read 50

- "`-r -n`", no initial write pass, no data comparison.



General | Journal | I/O Payload | I/O Behavior | Advanced I/O | Patterns | Comr

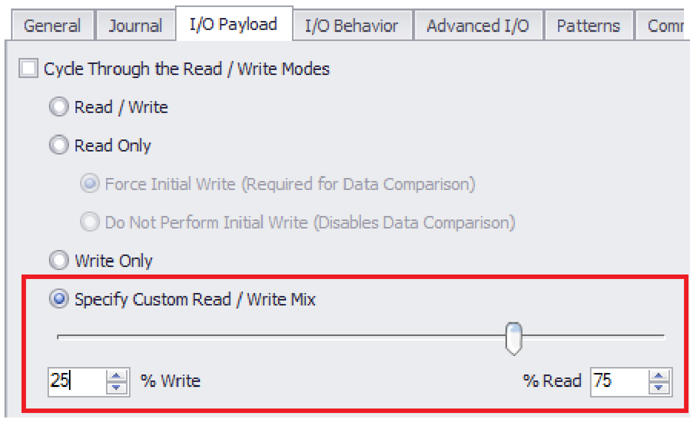
☐ Cycle Through the Read / Write Modes

Read / Write
[✓ Read Only]
Force Initial Write (Required for Data Comparison)
[✓ Do Not Perform Initial Write (Disables Data Comparison)]
Write Only
Specify Custom Read / Write Mix

50 % Write    % Read 50

- "`-ro`", initial write pass without data comparison

- Simplest form of randomized read/write mix (“`-%r`”, “`-%w`”)



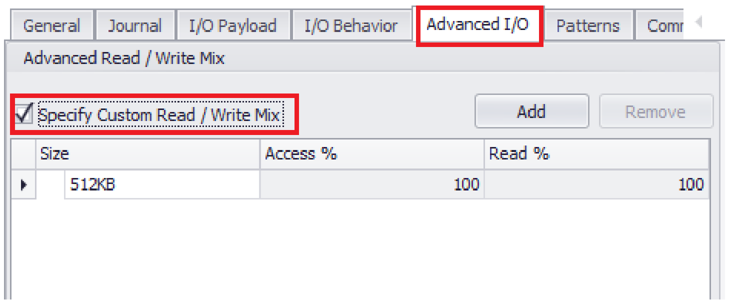
☐ Cycle Through the Read / Write Modes
- ○ Read / Write
- ○ Read Only
  - ● Force Initial Write (Required for Data Comparison)
  - ○ Do Not Perform Initial Write (Disables Data Comparison)
- ○ Write Only
- ● Specify Custom Read / Write Mix

25 % Write    % Read 75

**NOTE**

The GUI always presents the “weights” in terms of percentages.

- The full “`-%r`” and “`-%w`” specifiers are accessed in the “Advanced I/O” tab.



Advanced Read / Write Mix

☑ Specify Custom Read / Write Mix

| Size | Access % | Read % |
|---|---|---|
| 512KB | 100 | 100 |

## 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.

## 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 VIAVI traffic generator products add some records into the data portion of the traffic. This is the case for the 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 VIAVI's Xgig Analyzer TraceView application under the View menu, Decode Switches, VIAVI Signatures:

- **Don't Decode VIAVI Signatures**
  - This option disables the decoding of the VIAVI 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 189 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 189 for information on specifying the timestamp addition to the I/O signature in milliseconds using the `-Um` command.

2

# Using the Graphical User Interface

This chapter describes the Graphical User Interface (GUI) used by Medusa Labs Test Tools Suite. The topics discussed in this chapter are as follows:

## 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**

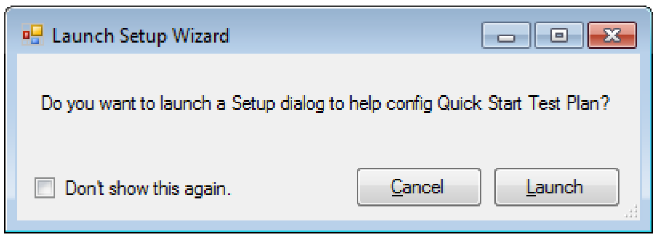
For Windows 8, select the **Start** button and the select the **Medusa Labs Test Tools** icon.

If this is the first time launching the GUI after installing the Medusa Labs Test Tools, the Launch Setup Wizard dialog box opens (see Quick Start section below). If this dialog box is later disabled, subsequent launch opens the Medusa Labs Test Tools main window.

#### Quick Start Overview

When you start the GUI, the “Launch Setup Wizard” dialog box is opened, unless the “Don’t show this again” option has been checked.

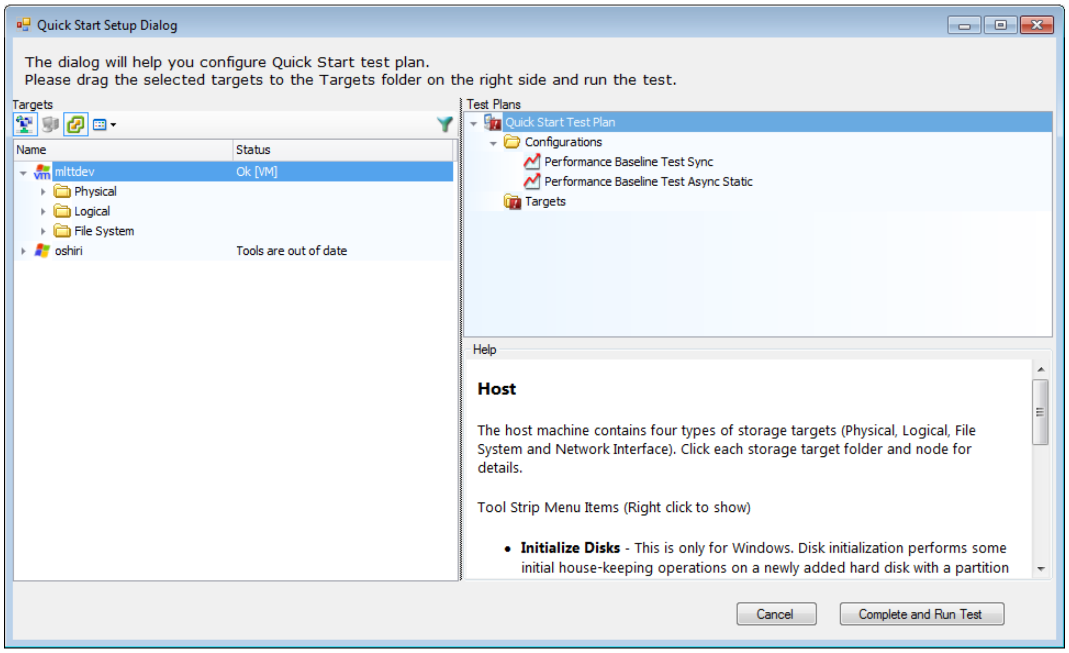
**Figure 7** Launch Setup Wizard


Launch Setup Wizard

Do you want to launch a Setup dialog to help config Quick Start Test Plan?

If you click the “Launch” button, it launches the “Quick Start Setup Dialog Box” (see Figure 8).

**Figure 8** Quick Start Setup Dialog box



The dialog will help you configure Quick Start test plan.
Please drag the selected targets to the Targets folder on the right side and run the test.

| Name | Status |
|---|---|
| mlttdev | Ok [VM] |
| Physical | |
| Logical | |
| File System | |
| oshiri | Tools are out of date |

- Quick Start Test Plan
  - Configurations
    - Performance Baseline Test Sync
    - Performance Baseline Test Async Static
  - Targets

**Host**

The host machine contains four types of storage targets (Physical, Logical, File System and Network Interface). Click each storage target folder and node for details.

Tool Strip Menu Items (Right click to show)

- **Initialize Disks** - This is only for Windows. Disk initialization performs some initial house-keeping operations on a newly added hard disk with a partition

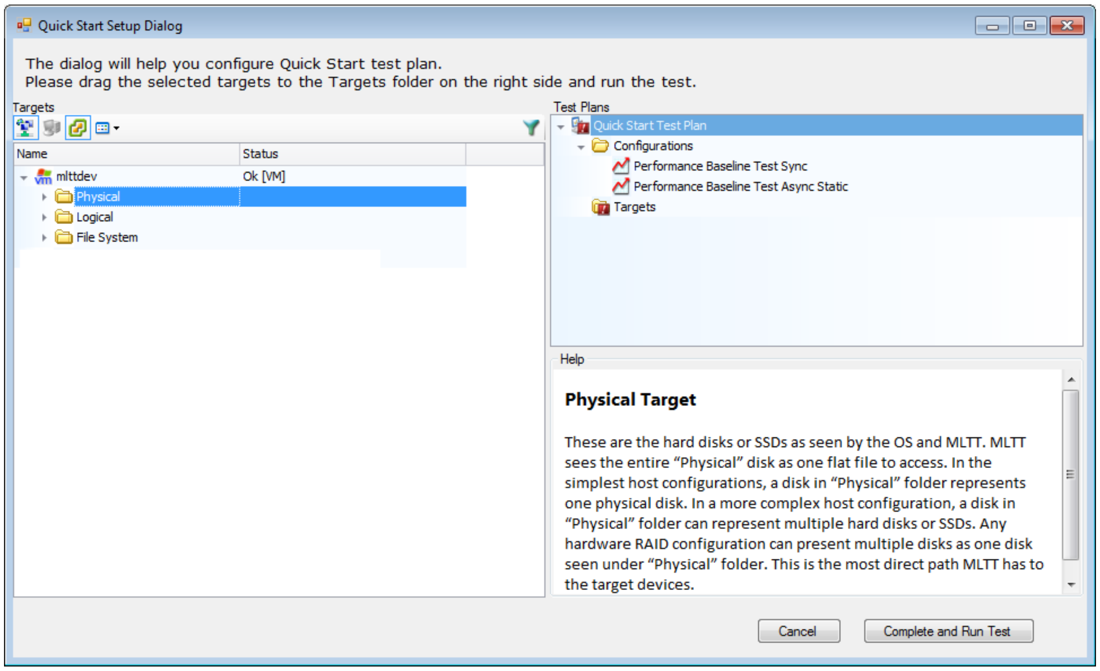
The “Targets” pane on the left has most of the functionalities of the “Targets” pane in the full GUI. The “Test Plans” on the upper right corner has the same functions as the “Test Plans” pane of the full GUI. However, it has following limitations:

- The “Test Plans” pane is populated with the “Quick Start Test Plan”. You cannot delete this plan or add new plans to this pane.
- The plan’s “Configurations” are populated with the two configurations shown. You cannot delete or edit these configurations, and you cannot add new configurations.

You can drag-and-drop targets from the “Targets” pane into the “Targets” folder. The “Help” pane on the lower right corner provides information on the “Targets” pane. The example above shows information about a host.

When you highlight a target-type folder, the “Help” pane displays the appropriate info for the selected target type (see Figure 9):

**Figure 9** Quick Start Setup Dialog box Help Pane



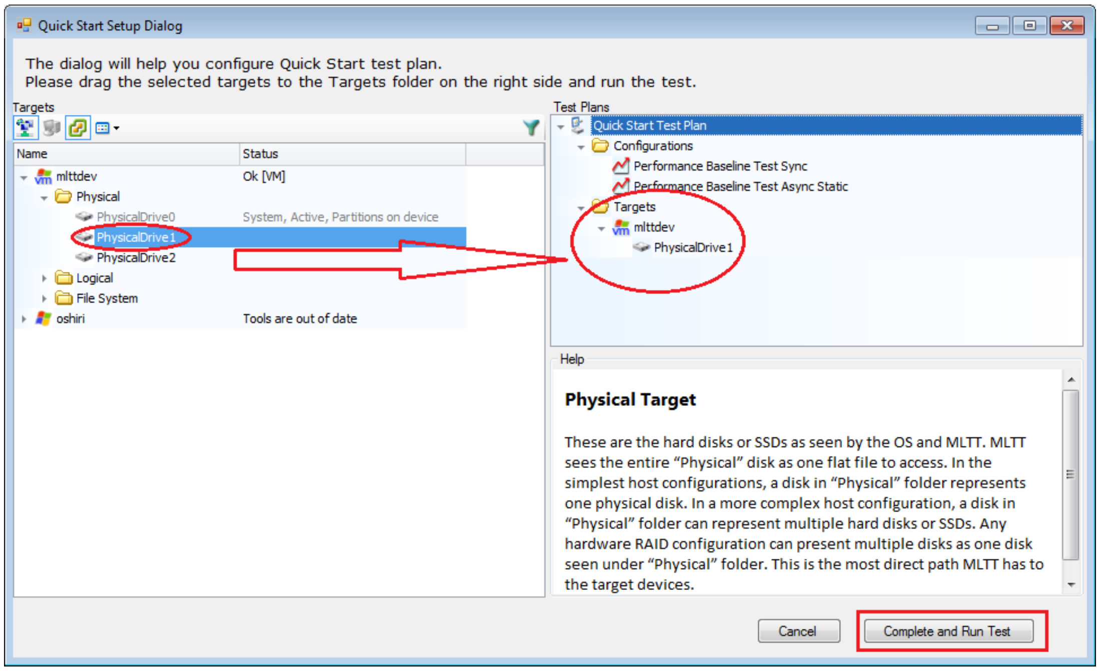
The dialog will help you configure Quick Start test plan.
Please drag the selected targets to the Targets folder on the right side and run the test.

**Physical Target**

These are the hard disks or SSDs as seen by the OS and MLTT. MLTT sees the entire “Physical” disk as one flat file to access. In the simplest host configurations, a disk in “Physical” folder represents one physical disk. In a more complex host configuration, a disk in “Physical” folder can represent multiple hard disks or SSDs. Any hardware RAID configuration can present multiple disks as one disk seen under “Physical” folder. This is the most direct path MLTT has to the target devices.

After you drag-n-drop one or more targets to the “Quick Start Test Plan”, push “Complete and Run Test” to run (see Figure 10).

**Figure 10** Quick Start Test Plan Complete and Run Test



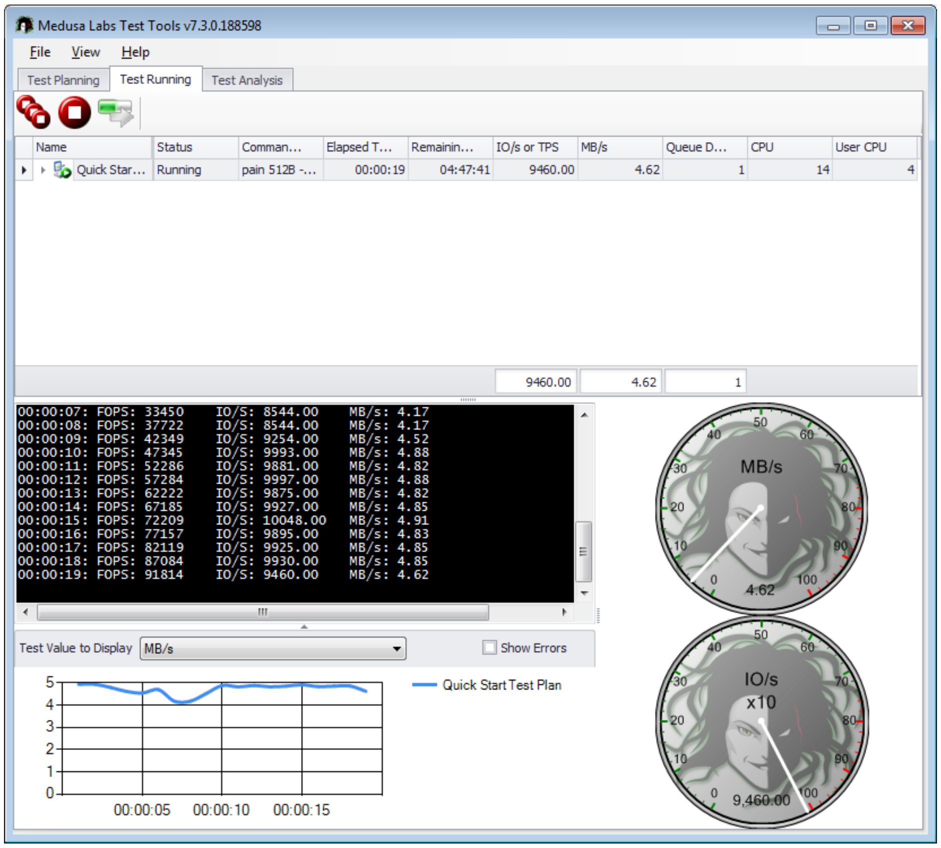
The dialog will help you configure Quick Start test plan.
Please drag the selected targets to the Targets folder on the right side and run the test.

**Physical Target**

These are the hard disks or SSDs as seen by the OS and MLTT. MLTT sees the entire “Physical” disk as one flat file to access. In the simplest host configurations, a disk in “Physical” folder represents one physical disk. In a more complex host configuration, a disk in “Physical” folder can represent multiple hard disks or SSDs. Any hardware RAID configuration can present multiple disks as one disk seen under “Physical” folder. This is the most direct path MLTT has to the target devices.

This closes the “Quick Start Setup Dialog” and launches the “Quick Start Test Plan” in the regular GUI (see Figure 11).

**Figure 11** Quick Start Test Plan Running

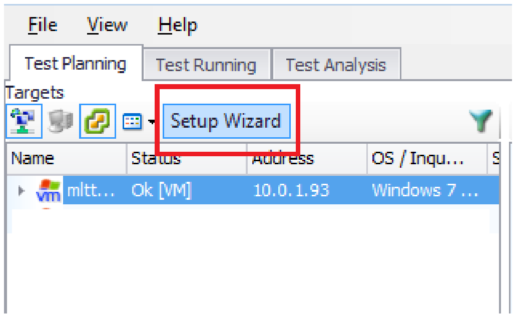


| Name | Status | Comman… | Elapsed T… | Remainin… | IO/s or TPS | MB/s | Queue D… | CPU | User CPU |
|---|---|---|---|---|---|---|---|---|---|
| Quick Star… | Running | pain 512B -… | 00:00:19 | 04:47:41 | 9460.00 | 4.62 | 1 | 14 | 4 |
| | | | | | 9460.00 | 4.62 | 1 | | |

```
00:00:07: FOPS: 33450    IO/S: 8544.00    MB/s: 4.17
00:00:08: FOPS: 37722    IO/S: 8544.00    MB/s: 4.17
00:00:09: FOPS: 42349    IO/S: 9254.00    MB/s: 4.52
00:00:10: FOPS: 47345    IO/S: 9993.00    MB/s: 4.88
00:00:11: FOPS: 52286    IO/S: 9881.00    MB/s: 4.82
00:00:12: FOPS: 57284    IO/S: 9997.00    MB/s: 4.88
00:00:13: FOPS: 62222    IO/S: 9875.00    MB/s: 4.82
00:00:14: FOPS: 67185    IO/S: 9927.00    MB/s: 4.85
00:00:15: FOPS: 72209    IO/S: 10048.00   MB/s: 4.91
00:00:16: FOPS: 77157    IO/S: 9895.00    MB/s: 4.83
00:00:17: FOPS: 82119    IO/S: 9925.00    MB/s: 4.85
00:00:18: FOPS: 87084    IO/S: 9930.00    MB/s: 4.85
00:00:19: FOPS: 91814    IO/S: 9460.00    MB/s: 4.62
```

The "Quick Start Setup Dialog" box can be opened any time from the full GUI as well even after you disable it to pop-up during the GUI start-up. Click "Setup Wizard" in the "Targets" pane tool bar in the full GUI (see Figure 12).

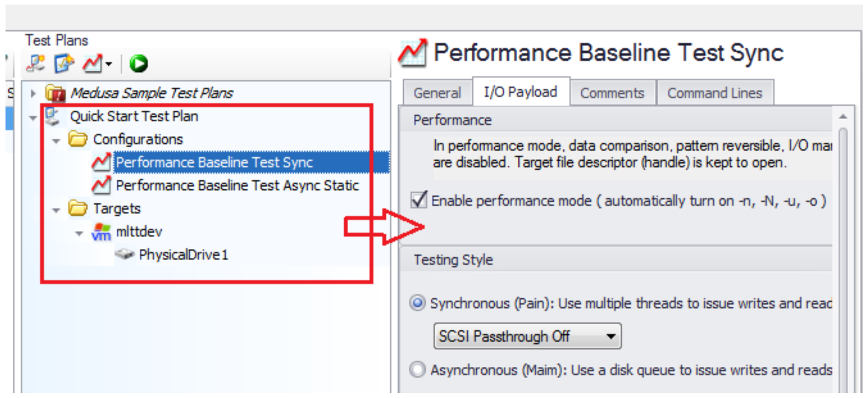
**Figure 12** Setup Wizard Launch



| Name | Status | Address | OS / Inqu… |
|---|---|---|---|
| mltt… | Ok [VM] | 10.0.1.93 | Windows 7 … |

"Quick Start Test Plan" instances launched from the "Quick Start Setup Dialog" box become resident in the "Test Plans" pane of the full GUI. Here, they can be edited just like any other Test Plan that was created in the full GUI.

**Figure 13** Quick Start Test Plan Example



In performance mode, data comparison, pattern reversible, I/O mai
are disabled. Target file descriptor (handle) is kept to open.

☑ Enable performance mode ( automatically turn on -n, -N, -u, -o )

● Synchronous (Pain): Use multiple threads to issue writes and reac

SCSI Passthrough Off

○ Asynchronous (Maim): Use a disk queue to issue writes and reads

## 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" 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 60
- "Test Planning Tab" on page 65
- "Test Running Tab" on page 84
- "Test Analysis Tab" on page 89

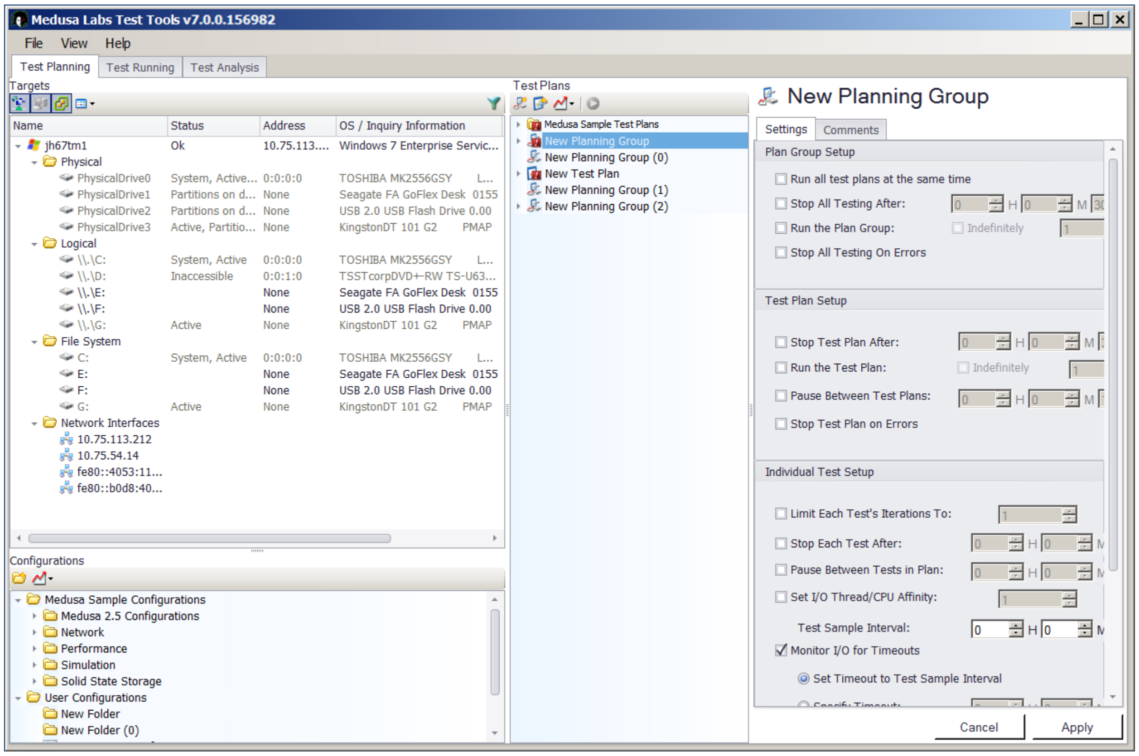
Refer to "GUI Overview" for a brief overview of the GUI.

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 14** Medusa Labs Test Tools GUI Window


Medusa Labs Test Tools v7.0.0.156982

Test Planning | Test Running | Test Analysis

Targets

| Name | Status | Address | OS / Inquiry Information |
|---|---|---|---|
| jh67tm1 | Ok | 10.75.113.... | Windows 7 Enterprise Servic... |
| Physical | | | |
| PhysicalDrive0 | System, Active... | 0:0:0:0 | TOSHIBA MK2556GSY L... |
| PhysicalDrive1 | Partitions on d... | None | Seagate FA GoFlex Desk 0155 |
| PhysicalDrive2 | Partitions on d... | None | USB 2.0 USB Flash Drive 0.00 |
| PhysicalDrive3 | Active, Partitio... | None | KingstonDT 101 G2 PMAP |
| Logical | | | |
| \\.\C: | System, Active | 0:0:0:0 | TOSHIBA MK2556GSY L... |
| \\.\D: | Inaccessible | 0:0:1:0 | TSSTcorpDVD+-RW TS-U63... |
| \\.\E: | | None | Seagate FA GoFlex Desk 0155 |
| \\.\F: | | None | USB 2.0 USB Flash Drive 0.00 |
| \\.\G: | Active | None | KingstonDT 101 G2 PMAP |
| File System | | | |
| C: | System, Active | 0:0:0:0 | TOSHIBA MK2556GSY L... |
| E: | | None | Seagate FA GoFlex Desk 0155 |
| F: | | None | USB 2.0 USB Flash Drive 0.00 |
| G: | Active | None | KingstonDT 101 G2 PMAP |
| Network Interfaces | | | |
| 10.75.113.212 | | | |
| 10.75.54.14 | | | |
| fe80::4053:11... | | | |
| fe80::b0d8:40... | | | |

Configurations

- Medusa Sample Configurations
  - Medusa 2.5 Configurations
  - Network
  - Performance
  - Simulation
  - Solid State Storage
- User Configurations
  - New Folder
  - New Folder (0)

Test Plans

- Medusa Sample Test Plans
- New Planning Group
- New Planning Group (0)
- New Test Plan
- New Planning Group (1)
- New Planning Group (2)

New Planning Group

Settings | Comments

Plan Group Setup

- ☐ Run all test plans at the same time
- ☐ Stop All Testing After: 0 H 0 M 30
- ☐ Run the Plan Group: ☐ Indefinitely 1
- ☐ Stop All Testing On Errors

Test Plan Setup

- ☐ Stop Test Plan After: 0 H 0 M
- ☐ Run the Test Plan: ☐ Indefinitely 1
- ☐ Pause Between Test Plans: 0 H 0 M
- ☐ Stop Test Plan on Errors

Individual Test Setup

- ☐ Limit Each Test's Iterations To: 1
- ☐ Stop Each Test After: 0 H 0 M
- ☐ Pause Between Tests in Plan: 0 H 0 M
- ☐ Set I/O Thread/CPU Affinity: 1
- Test Sample Interval: 0 H 0 M
- ✓ Monitor I/O for Timeouts
  - ● Set Timeout to Test Sample Interval

Cancel | Apply

The Medusa Labs Test Tools Main window contains the following components (Figure 14):

## Medusa Labs Test Tools Menu Bar

The **Menu** bar contains the **File**, **View**, and **Help** menu items. For more information about the MLTT menus, see "Medusa Labs Test Tools Menus" on page 60.

## Test Planning Tab

The "Test Planning Tab" on page 65 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

The "Test Running Tab" on page 84 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

The "Test Analysis Tab" on page 89 contains the following sections:

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

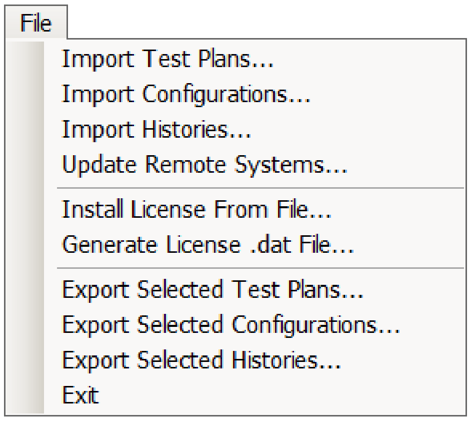
## Medusa Labs Test Tools Menus

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

### File Menu

The **File** menu (see Figure 15) 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 15** File Menu


File
Import Test Plans...
Import Configurations...
Import Histories...
Update Remote Systems...
Install License From File...
Generate License .dat File...
Export Selected Test Plans...
Export Selected Configurations...
Export Selected Histories...
Exit

#### 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 system ID (.dat) 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.

It is also possible to export test plans as a Python, Bash, or PowerShell script. In order to change what type of script you save the test plan as, change the **Save as type:** drop down box to select the option that you want.

**NOTE**
When exporting to a script, targets in the Test Plan are ignored. You must specify the targets when running the created scripts. Targets are specified by using target specifying switches that are described below.

The exported scripts take the following command line options.

**--targets <target file>:** Use this option to specify one or more targets for all Test Plans included in the script. The "target file" must be in the same format as the "targets.dat" target list file for pain/maim command line tools.

**--workdir <directory>:** Use this option to specify the script's base working directory. The tests will create the log files inside sub-directories created within this base working directory. The sub-directories are named according to the Test Plan and/or Planning Group names and time stamps similar to the directory structure created by the GUI. This switch is not mandatory. If not specified, the script's current working directory is the base working directory.

**--plan<Test Plan Number>-targets <Target File>:** Use this option to specify targets for a specific Test Plan in the script. The "Test Plan Number" is the ordinal number of a Test Plan within the Planning Group that was exported.

### 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.

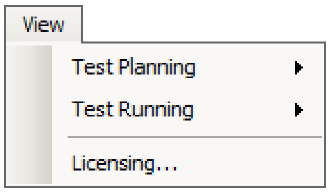
### Exit

Exits Medusa Labs Test Tools.

## View Menu

The **View** menu (see Figure 16) gives you the option to show or hide sections of the GUI main window. Options for the View menu are:

**Figure 16** 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.

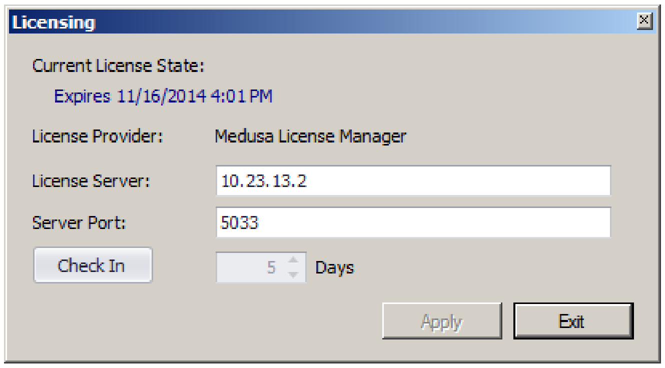
## Testing Running

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 17** Licensing Dialog



Licensing

Current License State:
Expires 11/16/2014 4:01 PM

License Provider: Medusa License Manager

License Server: 10.23.13.2

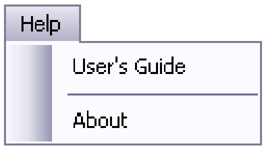
Server Port: 5033

**NOTE**
The Licensing dialog box for any system may be accessed. Refer to "Accessing System Licensing Information" on page 69 for instructions.

## Help Menu

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

**Figure 18** 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” on page 65 lists the targets available for testing with MLTT.

“Configurations Area” on page 70 is used to set up or select the configuration to use for testing.

“Test Plans Area” on page 71 is used to set up and select the test plan.

## Targets Area

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

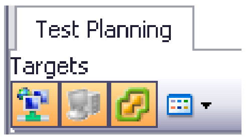
The target can be a file, logical drive, or physical drive that resides in the host system or is externally attached via SCSI, USB, FireWire, LAN, SAN, Sockets, and others.

The **Targets** area contains the following:

### Targets View Buttons

The Targets View buttons (shown in Figure 19) allow you to choose the type of targets to show in the **Targets Categories** section. A description of each button is also provided.

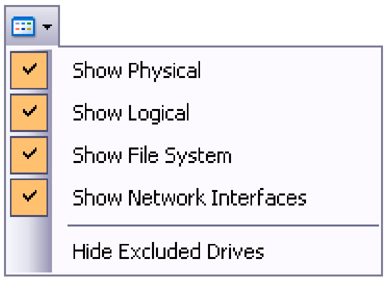
**Figure 19** Targets View Buttons



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.

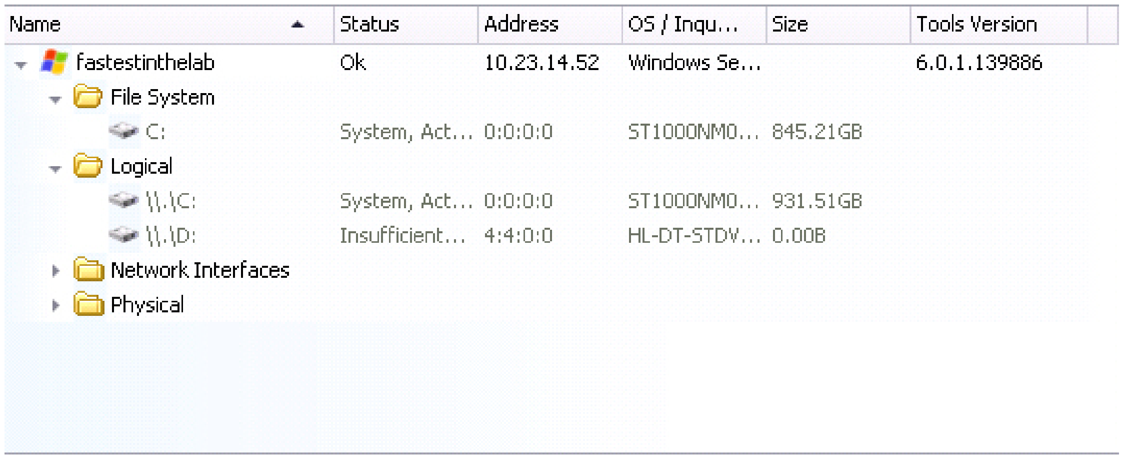


From the drop-down menu, you can show or hide physical drives, logical drives, and file systems that reside in the host system displayed in the **Target Categories** section. Targets can also be remote systems or systems that are externally attached via SCSI, USB, FireWire, LAN, SAN, and others.

## 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 20** Target Categories Section



| Name | Status | Address | OS / Inqu... | Size | Tools Version |
|---|---|---|---|---|---|
| fastestinthelab | Ok | 10.23.14.52 | Windows Se... | | 6.0.1.139886 |
| File System | | | | | |
| C: | System, Act... | 0:0:0:0 | ST1000NM0... | 845.21GB | |
| Logical | | | | | |
| \\.\C: | System, Act... | 0:0:0:0 | ST1000NM0... | 931.51GB | |
| \\.\D: | Insufficient... | 4:4:0:0 | HL-DT-STDV... | 0.00B | |
| Network Interfaces | | | | | |
| Physical | | | | | |

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.

### Storage Target Types

#### Physical

This is a “disk” as presented by an HBA to the OS. The actual topology below the HBA does not matter to MLTT. For example, it could be a VLUN within an actual physical disk configured before the OS boots or it could be any number of hardware RAID configurations involving several actual physical disks.

□There may be multi-path configured.

It may not be a physical disk, but instead, it could be a regular file on an NTFS file system on a remote system presented as a “disk” (such as, HBA emulation, software iSCSI client, etc.) It can be thought of as a “disk” is one big flat “file” that can be accessed by MLTT.

#### Logical

This is a “volume” created by the operating system on top of one or more “disks”. It may have RAID-like characteristics, but the constructs are above the HBA (that is, it’s all software”).

Mirrored

Spanning

Play around in Windows Disk Management

As with a “disk”, a “volume” is one big flat “file” that can be accessed by MLTT.

**File System**

A file system “target” is just a regular file residing in a formatted file system. For example, one that you can see the “target” file in Windows Explorer or with a “dir” command. It can be a network file system, such as CIFS (aka “SMB”, “Windows File Sharing”), NFS, or a way to test NAS devices which are generally mounted as network shares.

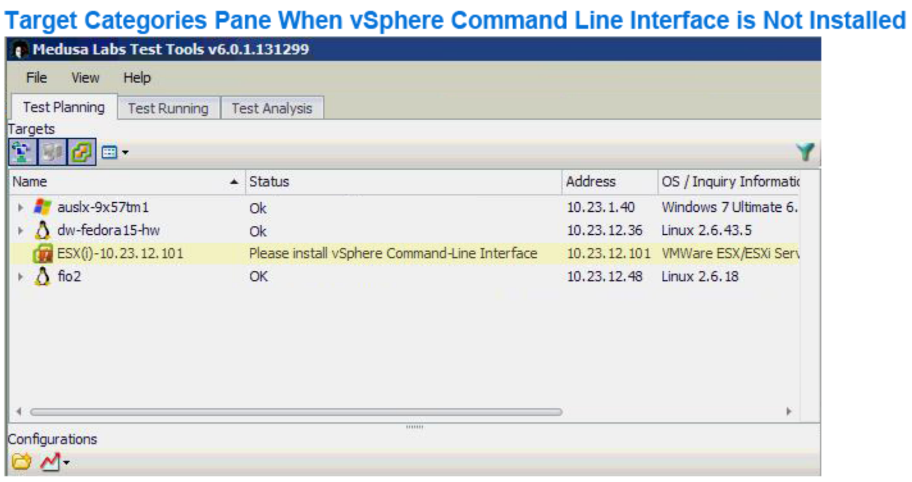
**NOTE**

If the specified file system target is a directory, pain/maim will automatically append the path separator (‘/’ or ‘\’, depending on the operating system) if necessary and append “targetfile.dat” as the file name.

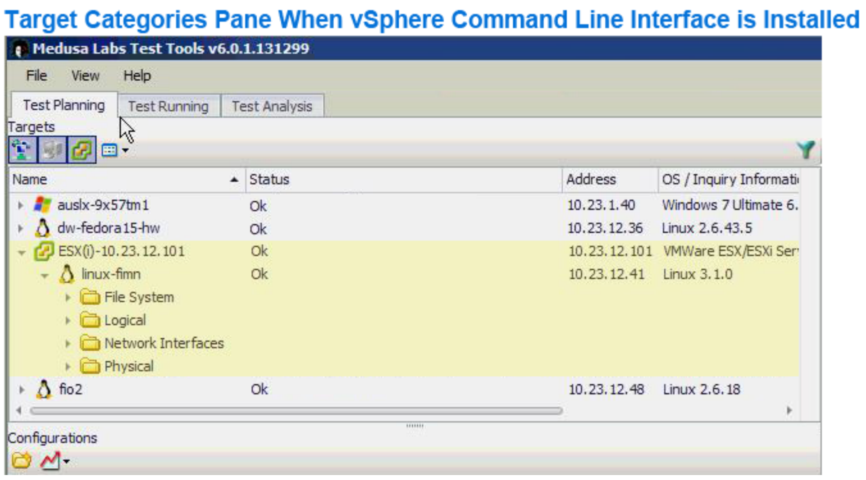
Some grayed-out targets cannot be selected because they have a target exclusion. These exclusions correspond with the ones used by catapult. This protection mechanism keeps critical data from being overwritten during testing. To override device exclusions for these targets, right-click on the target 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 21 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 21** vSphere Command Line Interface


Target Categories Pane When vSphere Command Line Interface is Not Installed
Medusa Labs Test Tools v6.0.1.131299
File
View
Help
Test Planning
Test Running
Test Analysis
Targets
Configurations

| Name | Status | Address | OS / Inquiry Informati |
|---|---|---|---|
| auslx-9x57tm1 | Ok | 10.23.1.40 | Windows 7 Ultimate 6. |
| dw-fedora15-hw | Ok | 10.23.12.36 | Linux 2.6.43.5 |
| ESX(i)-10.23.12.101 | Please install vSphere Command-Line Interface | 10.23.12.101 | VMWare ESX/ESXi Serv |
| fio2 | OK | 10.23.12.48 | Linux 2.6.18 |

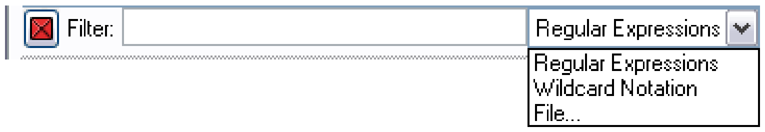

Target Categories Pane When vSphere Command Line Interface is Installed
Medusa Labs Test Tools v6.0.1.131299
File
View
Help
Test Planning
Test Running
Test Analysis
Targets
Configurations

| Name | Status | Address | OS / Inquiry Informati |
|---|---|---|---|
| auslx-9x57tm1 | Ok | 10.23.1.40 | Windows 7 Ultimate 6. |
| dw-fedora15-hw | Ok | 10.23.12.36 | Linux 2.6.43.5 |
| ESX(i)-10.23.12.101 | Ok | 10.23.12.101 | VMWare ESX/ESXi Ser |
| linux-fimn | Ok | 10.23.12.41 | Linux 3.1.0 |
| File System | | | |
| Logical | | | |
| Network Interfaces | | | |
| Physical | | | |
| fio2 | Ok | 10.23.12.48 | Linux 2.6.18 |

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 22 is displayed below the **Target Categories** section.

**Figure 22** 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 wild card 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** text box.

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.

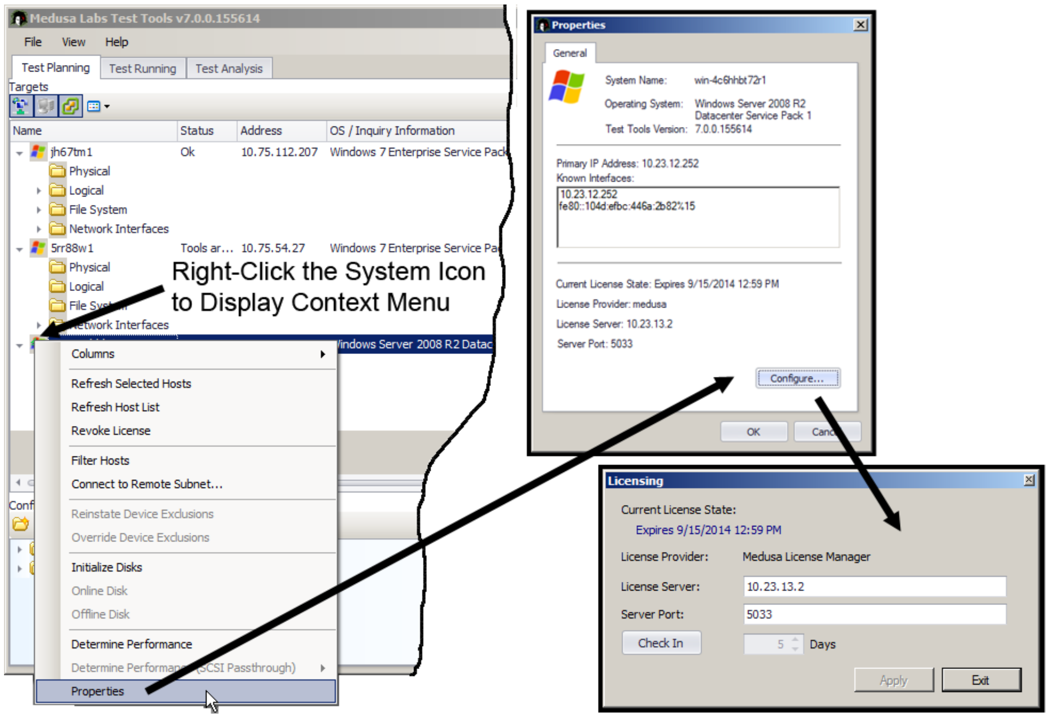
On versions of Windows 2003 R2 and later, devices can be brought online or offline. When devices are offline, they cannot be written to or read from and so are not good for testing. The GUI can bring drives online so that they can be tested.

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 23** Accessing System Licensing Information



| Name | Status | Address | OS / Inquiry Information |
|---|---|---|---|
| jh67tm1 | Ok | 10.75.112.207 | Windows 7 Enterprise Service Pack |
| Physical | | | |
| Logical | | | |
| File System | | | |
| Network Interfaces | | | |
| 5rr88w1 | Tools ar... | 10.75.54.27 | Windows 7 Enterprise Service Pa |
| Physical | | | |
| Logical | | | |
| File System | | | |
| Network Interfaces | | | |
| | | | Windows Server 2008 R2 Datac |

System Name: win-4c6hhbt72r1
Operating System: Windows Server 2008 R2 Datacenter Service Pack 1
Test Tools Version: 7.0.0.155614

Primary IP Address: 10.23.12.252
Known Interfaces:
10.23.12.252
fe80::104d:efbc:446a:2b82%15

Current License State: Expires 9/15/2014 12:59 PM
License Provider: medusa
License Server: 10.23.13.2
Server Port: 5033

Current License State:
Expires 9/15/2014 12:59 PM
License Provider: Medusa License Manager
License Server: 10.23.13.2
Server Port: 5033
5 Days

## 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".

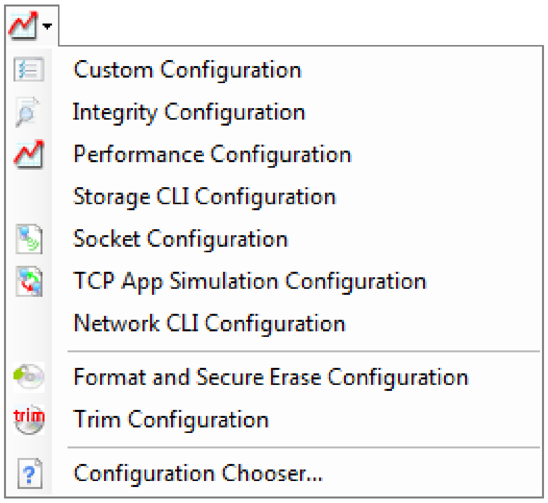
### New Folder Button

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 24** Create New Configuration Button


Custom Configuration
Integrity Configuration
Performance Configuration
Storage CLI Configuration
Socket Configuration
TCP App Simulation Configuration
Network CLI Configuration
Format and Secure Erase Configuration
Trim Configuration
Configuration Chooser...

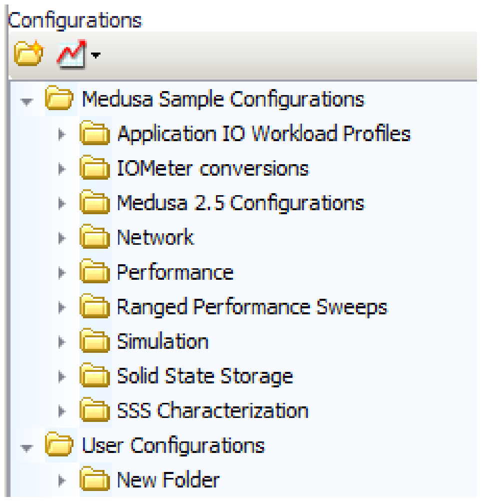
**NOTE**
The configuration editors are described in detail in Chapter 3 "Using the Configuration Editors".

You can also access the list of configurations by right-clicking in the Configurations pane and selecting **New Configuration**. Refer to Figure 31 on page 74 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 25** 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.

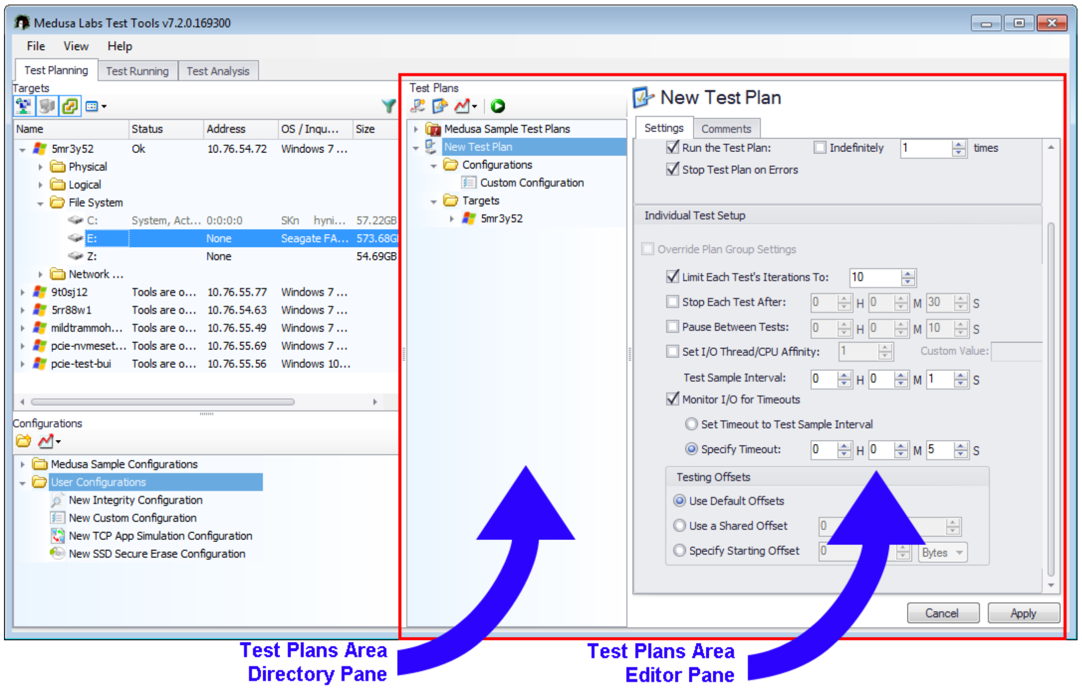
To rename any of the folders or configurations listed in the **User Configurations** folder, select the folder or configuration and click it again to edit the name.

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 72) 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 77).

**Figure 26** Test Plans Area



| Name | Status | Address | OS / Inqu... | Size |
|---|---|---|---|---|
| 5mr3y52 | Ok | 10.76.54.72 | Windows 7 ... | |
| Physical | | | | |
| Logical | | | | |
| File System | | | | |
| C: | System, Act... | 0:0:0:0 | SKn hyni... | 57.22GB |
| E: | | None | Seagate FA... | 573.68G |
| Z: | | None | | 54.69GB |
| Network ... | | | | |
| 9t0sj12 | Tools are o... | 10.76.55.77 | Windows 7 ... | |
| 5rr88w1 | Tools are o... | 10.76.54.63 | Windows 7 ... | |
| mildtrammoh... | Tools are o... | 10.76.55.49 | Windows 7 ... | |
| pcie-nvmeset... | Tools are o... | 10.76.55.69 | Windows 7 ... | |
| pcie-test-bui | Tools are o... | 10.76.55.56 | Windows 10... | |

Configurations
- Medusa Sample Configurations
- User Configurations
  - New Integrity Configuration
  - New Custom Configuration
  - New TCP App Simulation Configuration
  - New SSD Secure Erase Configuration

Test Plans
- Medusa Sample Test Plans
- New Test Plan
  - Configurations
    - Custom Configuration
  - Targets
    - 5mr3y52

New Test Plan

- [✓] Run the Test Plan: [ ] Indefinitely 1 times
- [✓] Stop Test Plan on Errors

Individual Test Setup

- [ ] Override Plan Group Settings
- [✓] Limit Each Test's Iterations To: 10
- [ ] Stop Each Test After: 0 H 0 M 30 S
- [ ] Pause Between Tests: 0 H 0 M 10 S
- [ ] Set I/O Thread/CPU Affinity: 1 Custom Value:
- Test Sample Interval: 0 H 0 M 1 S
- [✓] Monitor I/O for Timeouts
  - ( ) Set Timeout to Test Sample Interval
  - (●) Specify Timeout: 0 H 0 M 5 S

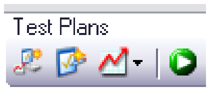
Testing Offsets

- (●) Use Default Offsets
- ( ) Use a Shared Offset 0
- ( ) Specify Starting Offset 0 Bytes

## Test Plans Directory Pane

The Test Plans directory pane consists of the four buttons near the top of the pane (shown in Figure 27) and the Test Plan Browser which displays all the available planning group and test plan objects.

**Figure 27** Directory Pane Buttons

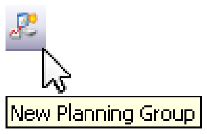


From left to right, these buttons are the **New Planning Group** button, the **New Test Plan** button, the **New Configuration** button, and the **Press start to begin tests** button.

### 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 28** New Planning Group Button



## New Test Plan Button

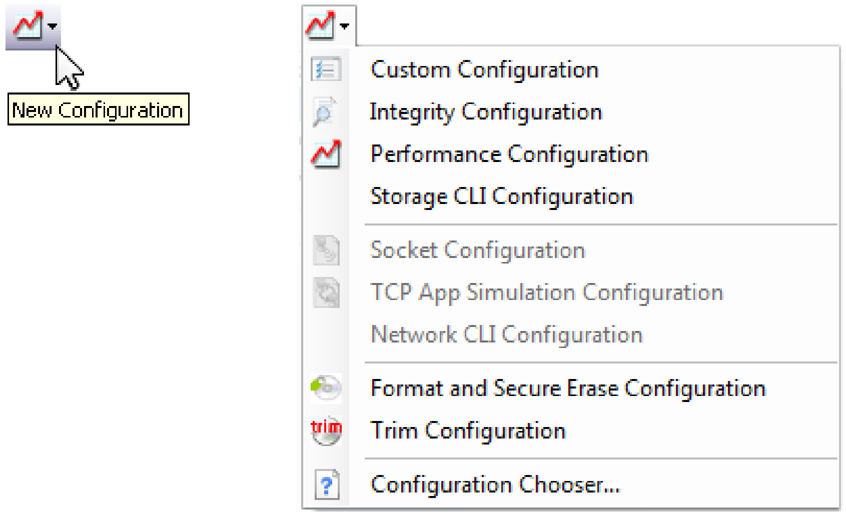
This button allows you to create a new test plan. When the button is selected, the new test plan is listed in the directory pane and the new test plan editor is displayed in the editor pane.

**Figure 29** 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 Format and Secure Erase, and the Trim configurations available.

**Figure 30** New Configuration Button

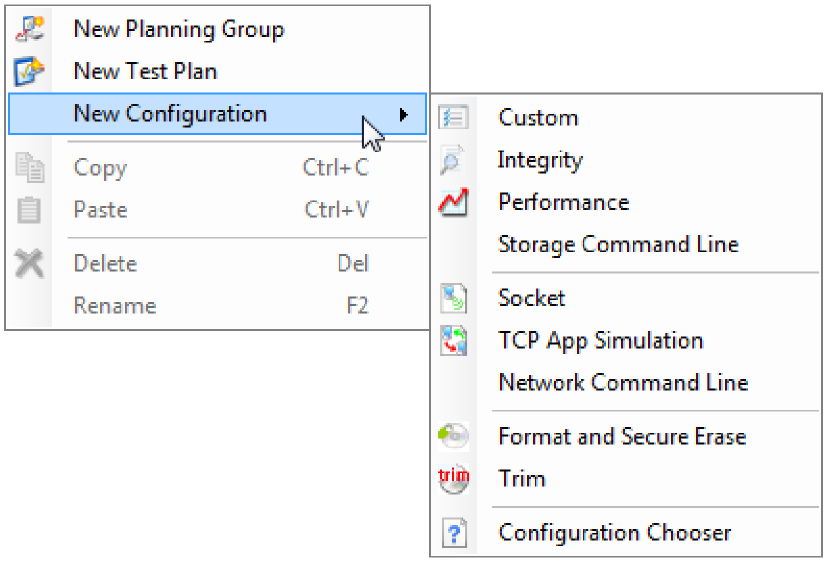

New Configuration
Custom Configuration
Integrity Configuration
Performance Configuration
Storage CLI Configuration
Socket Configuration
TCP App Simulation Configuration
Network CLI Configuration
Format and Secure Erase Configuration
trim
Trim Configuration
Configuration Chooser...

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”.

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**.

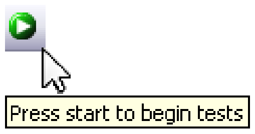
**Figure 31** Right Click Test Plans Pane



## Press start to begin tests Button

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

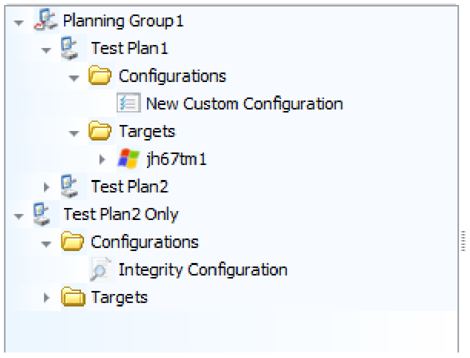
**Figure 32** Press start to begin tests Button



## Test Plan Browser

The **Test Plan Browser** (shown in Figure 33) displays all the planning groups, test plans, configurations, and targets that you created or copied to the directory pane.

**Figure 33** 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.

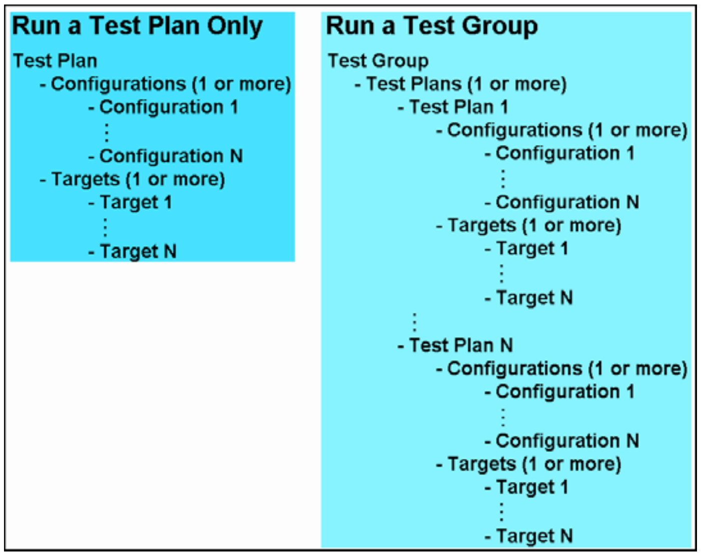
- When the icon is an arrow pointing right ( ▶ ), the objects are hidden; click the arrow to show the objects.
- When the icon is an arrow pointing down ( ▼ ), the objects are shown; click the arrow to hide the objects.

**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 34.

**Figure 34** 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 77.

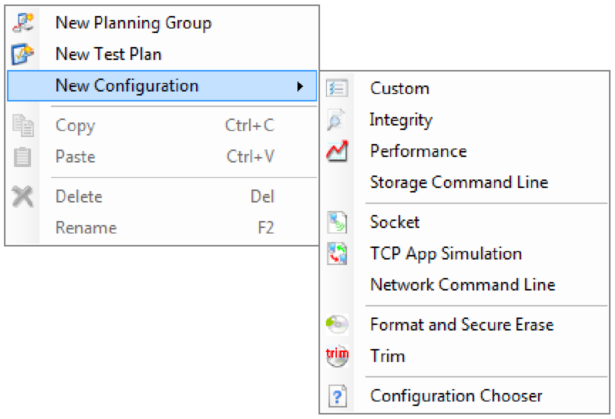
The Test Plan contains the configurations (1 or more) and the targets (1 or more). When the Test Plan is run, it will run all of its Configurations against all of its Targets. A Test Group containing the Test Plan is not required to run the one Test Plan only. You can edit the Test Plan in the Test Plans Area Editor Pane. Refer to "Test Plan Editor" on page 80.

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 83.

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 66. When selected, the target information is displayed in the Test Plans Area Editor Pane.

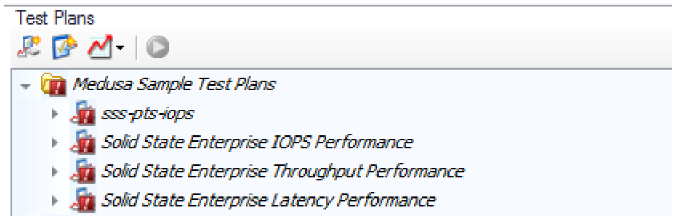
New planning groups, test plans, and configurations can also be added by right-clicking in the test plan browser to display the context menu shown in Figure 35.

**Figure 35** Test Plan Browser Context Menu



Also included in the Test Plans Browser Directory Pane is the Medusa Sample Test Plans folder see Figure 36. This folder provides test plans that have been created by the Medusa Labs team for you to copy and paste for specific testing purposes.

**Figure 36** Medusa Sample Test Plans


Test Plans
Medusa Sample Test Plans
sss-pts-iops
Solid State Enterprise IOPS Performance
Solid State Enterprise Throughput Performance
Solid State Enterprise Latency Performance

You can copy a test plan from this folder and then add it as a stand-alone test plan or to a Test Plan Group.

**NOTE**

You cannot modify or delete these sample test plans. These sample test plans may only be modified or edited after they have been copied and pasted into the Test Plans Browser Directory Pane.

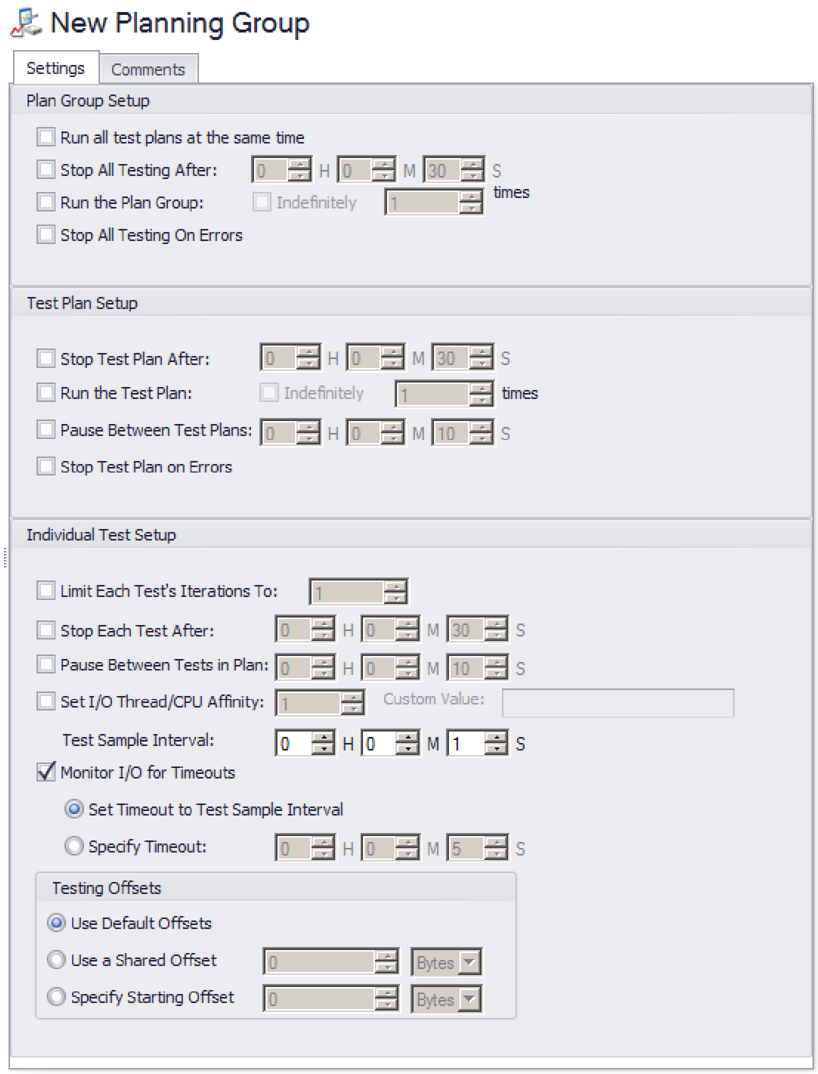
## Test Plans Editor Pane

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 80, or "Configuration Editor" on page 83.

### 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.) See Figure 37 for the editable **Planning Group** properties.

**Figure 37** Planning Group Editor



New Planning Group

Settings | Comments

Plan Group Setup

- [ ] Run all test plans at the same time
- [ ] Stop All Testing After: 0 H 0 M 30 S
- [ ] Run the Plan Group: [ ] Indefinitely 1 times
- [ ] Stop All Testing On Errors

Test Plan Setup

- [ ] Stop Test Plan After: 0 H 0 M 30 S
- [ ] Run the Test Plan: [ ] Indefinitely 1 times
- [ ] Pause Between Test Plans: 0 H 0 M 10 S
- [ ] Stop Test Plan on Errors

Individual Test Setup

- [ ] Limit Each Test's Iterations To: 1
- [ ] Stop Each Test After: 0 H 0 M 30 S
- [ ] Pause Between Tests in Plan: 0 H 0 M 10 S
- [ ] Set I/O Thread/CPU Affinity: 1 Custom Value:

Test Sample Interval: 0 H 0 M 1 S

- [✓] Monitor I/O for Timeouts

(●) Set Timeout to Test Sample Interval
( ) Specify Timeout: 0 H 0 M 5 S

Testing Offsets

(●) Use Default Offsets
( ) Use a Shared Offset 0 Bytes
( ) Specify Starting Offset 0 Bytes

## *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.

Note that iteration count is not applicable to any random access I/O because random access I/O will never complete a write or read of an entire file. Therefore, this setting is not useful when performing random access.

**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.

**Custom Value –** Enter the CPU numbers in this field to specify selected CPUs in a system. Use a comma "," to separate individual CPU numbers and a hyphen "-" to specify a range of CPU numbers. For example in a system with 2 quad-core CPUs (8 "logical" CPUs total), if you were to enter 1,4,6-8, CPUs numbered 1, 4, 6, 7, and 8 would be specified. When values are included in this field, the **Set I/O Thread/CPU Affinity** spinner value is rendered inactive.

**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 1MB 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*

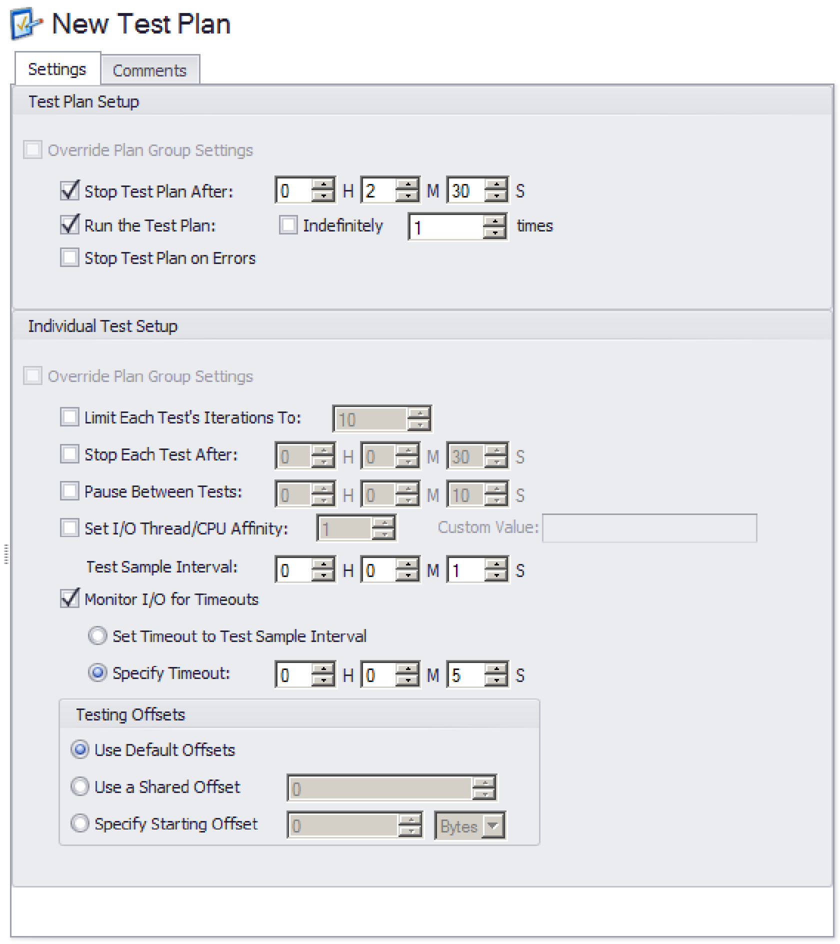
Enter your comments for the configuration in the comment box of the Comments tab.

**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.) See Figure 38 for the editable **Test Plan** properties.

**Figure 38** Test Plan Editor



New Test Plan

Settings | Comments

Test Plan Setup

☐ Override Plan Group Settings

[✓] Stop Test Plan After: 0 H 2 M 30 S

[✓] Run the Test Plan: ☐ Indefinitely 1 times

☐ Stop Test Plan on Errors

Individual Test Setup

☐ Override Plan Group Settings

☐ Limit Each Test's Iterations To: 10

☐ Stop Each Test After: 0 H 0 M 30 S

☐ Pause Between Tests: 0 H 0 M 10 S

☐ Set I/O Thread/CPU Affinity: 1 Custom Value:

Test Sample Interval: 0 H 0 M 1 S

[✓] Monitor I/O for Timeouts

○ Set Timeout to Test Sample Interval

● Specify Timeout: 0 H 0 M 5 S

Testing Offsets

● Use Default Offsets

○ Use a Shared Offset 0

○ Specify Starting Offset 0 Bytes

## *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.

Note that iteration count is not applicable to any random access I/O because random access I/O will never complete a write or read of an entire file. Therefore, this setting is not useful when performing random access.

**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.

**Custom Value –** Enter the CPU numbers in this field to specify selected CPUs in a system. Use a comma "`,`" to separate individual CPU numbers and a hyphen "`-`" to specify a range of CPU numbers. For example in a system with 2 quad-core CPUs (8 "logical" CPUs total), if you were to enter 1,4,6-8, CPUs numbered 1, 4, 6, 7, and 8 would be specified. When values are included in this field, the **Set I/O Thread/CPU Affinity** spinner value is rendered inactive.

**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.

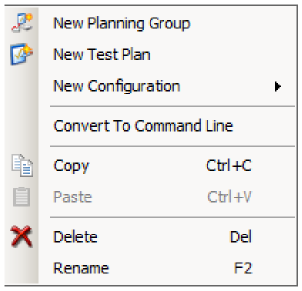
**Apply button –** Click this button when you are done editing the test plan properties.

## 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 (see Figure 39), when a configuration is right-clicked, the context menu displays a **Convert to Command Line** option.

**Figure 39** Convert to Command Line Option



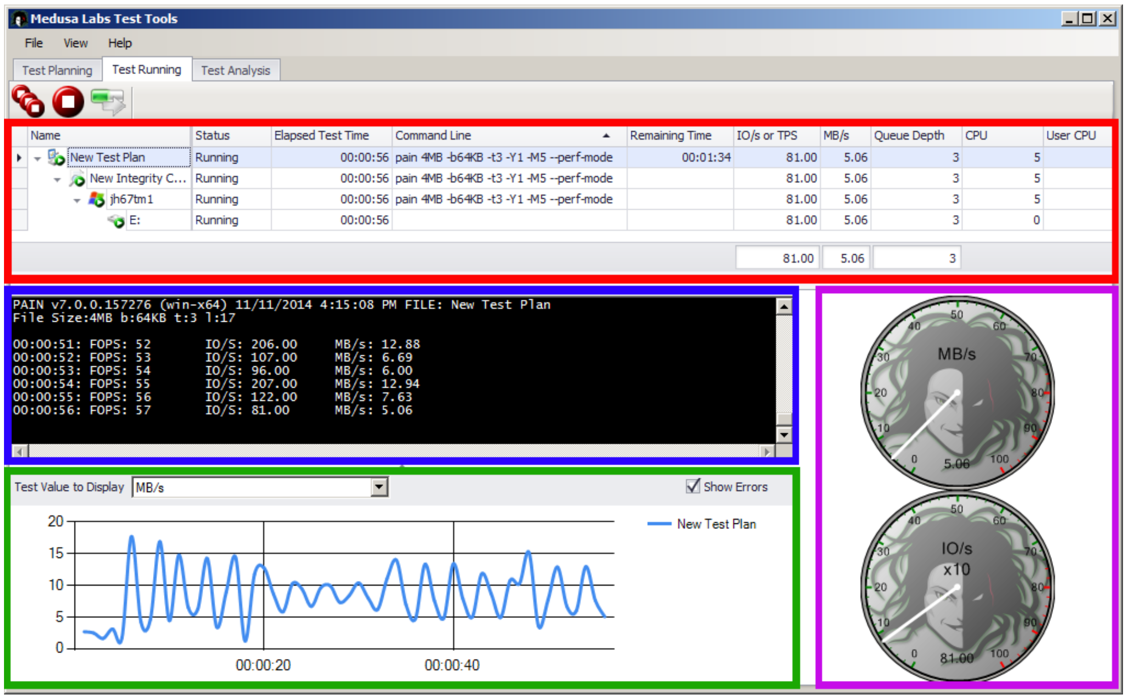
This option converts the configuration into the appropriate CLI configuration. Custom, Integrity, and Performance configurations are converted to Storage CLI configurations. Socket and TCP App Simulations are converted to Network CLI configurations. This option is not available for Storage CLI and Network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

These editors are described in detail in Chapter 3 “Using the Configuration Editors” starting on page 95. 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 40. All tests are removed as soon as they complete.

**Figure 40** Sample Text View of the Test Running Tab



| Name | Status | Elapsed Test Time | Command Line | Remaining Time | IO/s or TPS | MB/s | Queue Depth | CPU | User CPU |
|---|---|---|---|---|---|---|---|---|---|
| New Test Plan | Running | 00:00:56 | pain 4MB -b64KB -t3 -Y1 -M5 --perf-mode | 00:01:34 | 81.00 | 5.06 | 3 | 5 | |
| New Integrity C... | Running | 00:00:56 | pain 4MB -b64KB -t3 -Y1 -M5 --perf-mode | | 81.00 | 5.06 | 3 | 5 | |
| jh67tm1 | Running | 00:00:56 | pain 4MB -b64KB -t3 -Y1 -M5 --perf-mode | | 81.00 | 5.06 | 3 | 5 | |
| E: | Running | 00:00:56 | | | 81.00 | 5.06 | 3 | 0 | |
| | | | | | 81.00 | 5.06 | 3 | | |

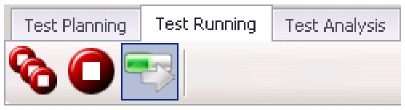
```
PAIN v7.0.0.157276 (win-x64) 11/11/2014 4:15:08 PM FILE: New Test Plan
File Size:4MB b:64KB t:3 l:17

00:00:51: FOPS: 52      IO/S: 206.00     MB/s: 12.88
00:00:52: FOPS: 53      IO/S: 107.00     MB/s: 6.69
00:00:53: FOPS: 54      IO/S: 96.00      MB/s: 6.00
00:00:54: FOPS: 55      IO/S: 207.00     MB/s: 12.94
00:00:55: FOPS: 56      IO/S: 122.00     MB/s: 7.63
00:00:56: FOPS: 57      IO/S: 81.00      MB/s: 5.06
```

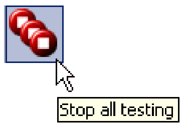
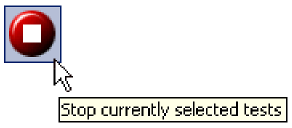
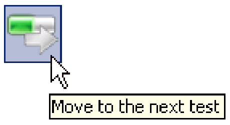
The **Test Running** tab has four panes:

- **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 41 and a description of each is also provided.

**Figure 41** 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.<br><br>When the Planning Group level is selected:<br><br>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:<br>– 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.<br>– 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
Hide All General Columns
Name
Status
Command Line
Elapsed Test Time
Remaining Time
Show All Testing Statistic Columns
Hide All Testing Statistic Columns
Avg I/O Completion/Response Time (Sec)
Avg IO/s or TPS
Avg MB/s
Avg Queue Depth
IO/s or TPS
MB/s
Queue Depth
Max I/O Completion/Response Time (Sec)
Max IO/s or TPS
Max MB/s
Max Queue Depth
Min I/O Completion/Response Time (Sec)
Min IO/s or TPS
Min MB/s
Min Queue Depth
CPU
User CPU
Show All Error Columns
Hide All Error Columns
Automatically Show Columns with Errors
Close Errors
Data Corruptions
Flush Errors
Initial Errors
I/O Halts
I/O Timeouts
License Errors
Open Errors
Read Errors
Remove Errors
Seek Errors
Size Errors
Startup Errors
Unknown Errors
Write Errors
Show All Final Statistic Columns
Hide All Final Statistic Columns
Total Bytes
Total Errors
Total File Operations
Total I/Os or Transactions

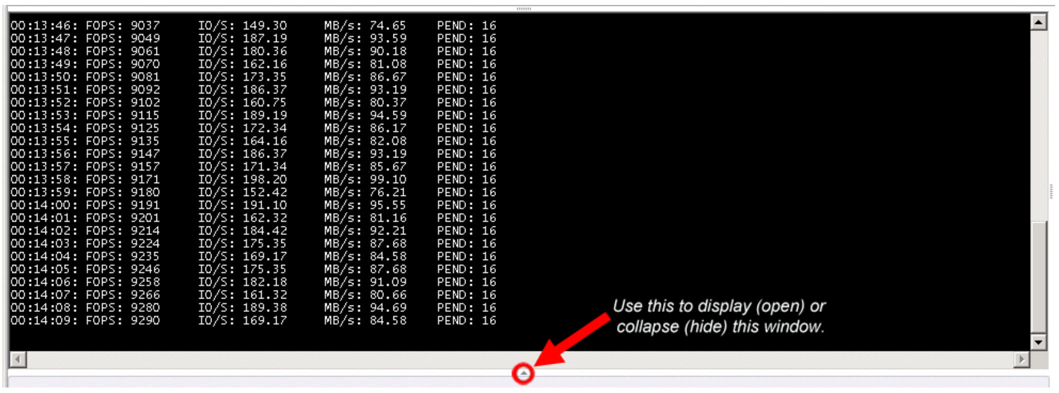
**NOTE**

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 42) 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 42** Text View Pane



```
00:13:46: FOPS: 9037     IO/S: 149.30     MB/s: 74.65     PEND: 16
00:13:47: FOPS: 9049     IO/S: 187.19     MB/s: 93.59     PEND: 16
00:13:48: FOPS: 9061     IO/S: 180.36     MB/s: 90.18     PEND: 16
00:13:49: FOPS: 9070     IO/S: 162.16     MB/s: 81.08     PEND: 16
00:13:50: FOPS: 9081     IO/S: 173.35     MB/s: 86.67     PEND: 16
00:13:51: FOPS: 9092     IO/S: 186.37     MB/s: 93.19     PEND: 16
00:13:52: FOPS: 9102     IO/S: 160.75     MB/s: 80.37     PEND: 16
00:13:53: FOPS: 9115     IO/S: 189.19     MB/s: 94.59     PEND: 16
00:13:54: FOPS: 9125     IO/S: 172.34     MB/s: 86.17     PEND: 16
00:13:55: FOPS: 9135     IO/S: 164.16     MB/s: 82.08     PEND: 16
00:13:56: FOPS: 9147     IO/S: 186.37     MB/s: 93.19     PEND: 16
00:13:57: FOPS: 9157     IO/S: 171.34     MB/s: 85.67     PEND: 16
00:13:58: FOPS: 9171     IO/S: 198.20     MB/s: 99.10     PEND: 16
00:13:59: FOPS: 9180     IO/S: 152.42     MB/s: 76.21     PEND: 16
00:14:00: FOPS: 9191     IO/S: 191.10     MB/s: 95.55     PEND: 16
00:14:01: FOPS: 9201     IO/S: 162.32     MB/s: 81.16     PEND: 16
00:14:02: FOPS: 9214     IO/S: 184.42     MB/s: 92.21     PEND: 16
00:14:03: FOPS: 9224     IO/S: 175.35     MB/s: 87.68     PEND: 16
00:14:04: FOPS: 9235     IO/S: 169.17     MB/s: 84.58     PEND: 16
00:14:05: FOPS: 9246     IO/S: 175.35     MB/s: 87.68     PEND: 16
00:14:06: FOPS: 9258     IO/S: 182.18     MB/s: 91.09     PEND: 16
00:14:07: FOPS: 9266     IO/S: 161.32     MB/s: 80.66     PEND: 16
00:14:08: FOPS: 9280     IO/S: 189.38     MB/s: 94.69     PEND: 16
00:14:09: FOPS: 9290     IO/S: 169.17     MB/s: 84.58     PEND: 16
```

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 43) displays the progression of the test at each sample interval. The sample interval was identified in the test plan setup.

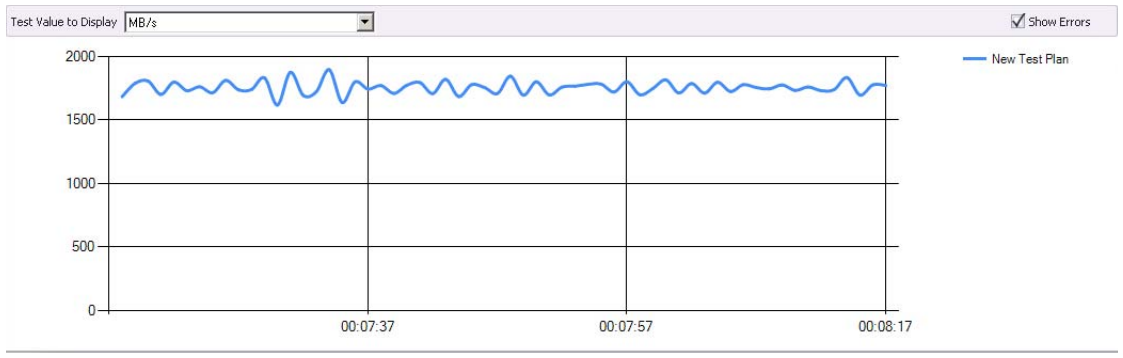
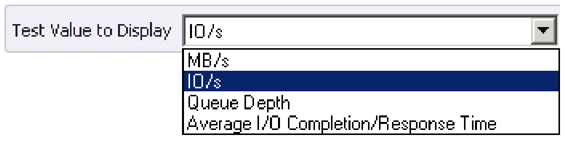


**Figure 43** Graph View Pane

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 drop-down list is shown in Figure 44. Changing this value changes the vertical scale on the graph.

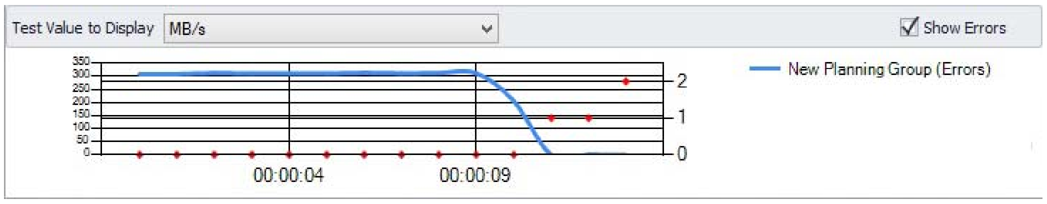
**Figure 44** Test Value to Display



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 45.
- 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.

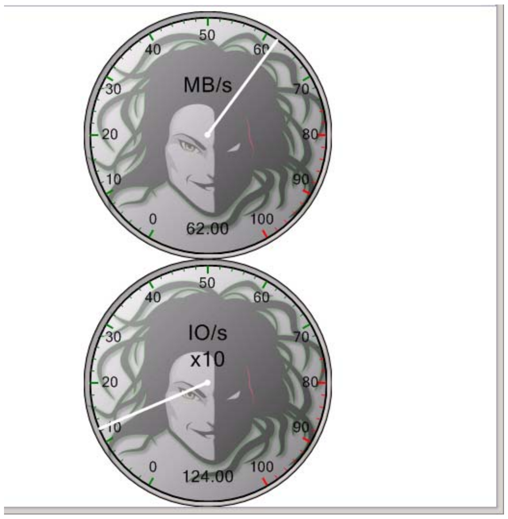
**Figure 45** Graph View Pane with Errors



## Speedometers Pane

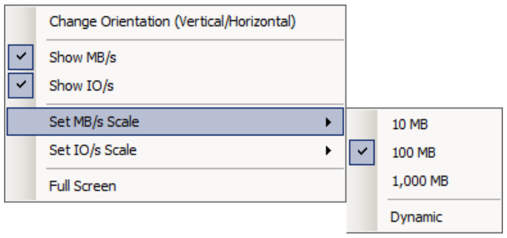
The **Speedometers** pane (shown in Figure 46) displays the real-time speed of the tests. There are two speedometers, one for MB/s and the other IO/s.

**Figure 46** 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 47.

**Figure 47** 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.

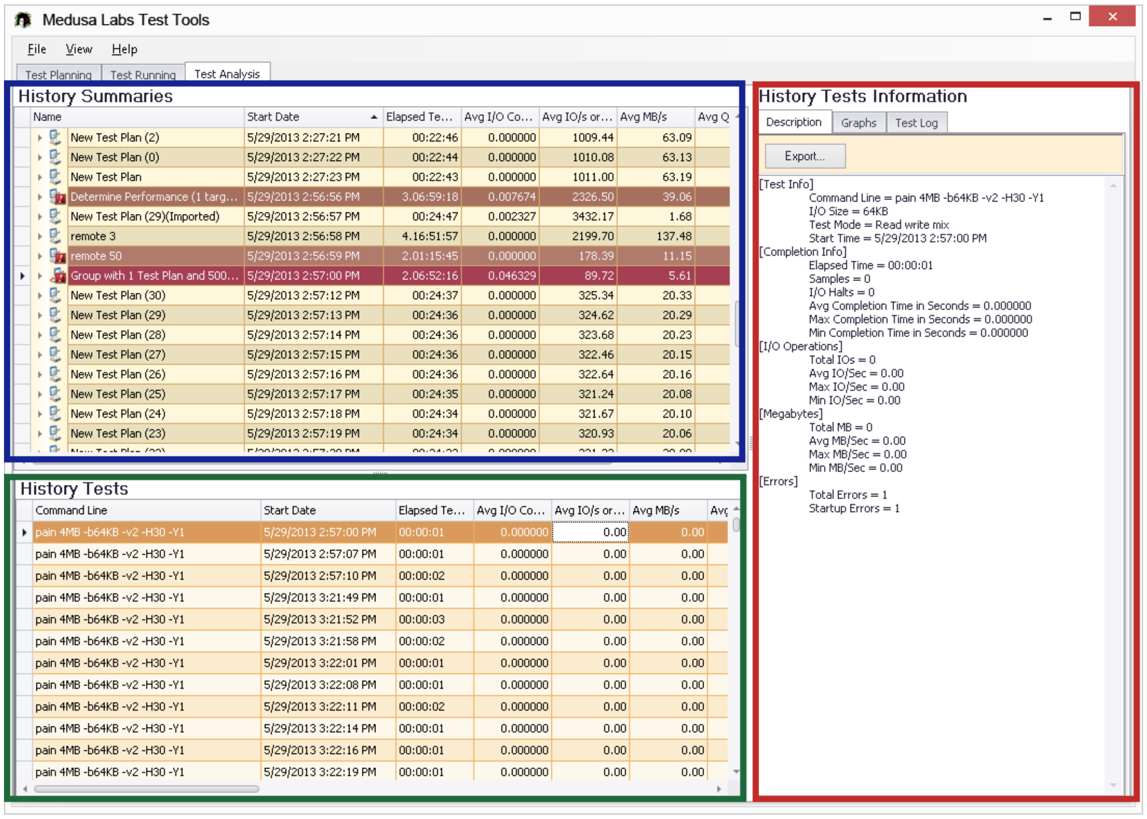
**Set IO/s Scale** – Select this option to choose the intervals displayed on the IO/s scale. You may select 100 IO/s, 1,000 IO/s, 10,000 IO/s, 100,000 IO/s, or Dynamic.

**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 48) displays the analysis of outputs of the tests that you have run.

**Figure 48** Sample Test Analysis Tab



History Summaries

| Name | Start Date | Elapsed Te... | Avg I/O Co... | Avg IO/s or... | Avg MB/s | Avg Q |
|---|---|---|---|---|---|---|
| New Test Plan (2) | 5/29/2013 2:27:21 PM | 00:22:46 | 0.000000 | 1009.44 | 63.09 | |
| New Test Plan (0) | 5/29/2013 2:27:22 PM | 00:22:44 | 0.000000 | 1010.08 | 63.13 | |
| New Test Plan | 5/29/2013 2:27:23 PM | 00:22:43 | 0.000000 | 1011.00 | 63.19 | |
| Determine Performance (1 targ... | 5/29/2013 2:56:56 PM | 3.06:59:18 | 0.007674 | 2326.50 | 39.06 | |
| New Test Plan (29)(Imported) | 5/29/2013 2:56:57 PM | 00:24:47 | 0.002327 | 3432.17 | 1.68 | |
| remote 3 | 5/29/2013 2:56:58 PM | 4.16:51:57 | 0.000000 | 2199.70 | 137.48 | |
| remote 50 | 5/29/2013 2:56:59 PM | 2.01:15:45 | 0.000000 | 178.39 | 11.15 | |
| Group with 1 Test Plan and 500... | 5/29/2013 2:57:00 PM | 2.06:52:16 | 0.046329 | 89.72 | 5.61 | |
| New Test Plan (30) | 5/29/2013 2:57:12 PM | 00:24:37 | 0.000000 | 325.34 | 20.33 | |
| New Test Plan (29) | 5/29/2013 2:57:13 PM | 00:24:36 | 0.000000 | 324.62 | 20.29 | |
| New Test Plan (28) | 5/29/2013 2:57:14 PM | 00:24:36 | 0.000000 | 323.68 | 20.23 | |
| New Test Plan (27) | 5/29/2013 2:57:15 PM | 00:24:36 | 0.000000 | 322.46 | 20.15 | |
| New Test Plan (26) | 5/29/2013 2:57:16 PM | 00:24:36 | 0.000000 | 322.64 | 20.16 | |
| New Test Plan (25) | 5/29/2013 2:57:17 PM | 00:24:35 | 0.000000 | 321.24 | 20.08 | |
| New Test Plan (24) | 5/29/2013 2:57:18 PM | 00:24:34 | 0.000000 | 321.67 | 20.10 | |
| New Test Plan (23) | 5/29/2013 2:57:19 PM | 00:24:34 | 0.000000 | 320.93 | 20.06 | |

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 | |

History Tests Information

```
[Test Info]
        Command Line = pain 4MB -b64KB -v2 -H30 -Y1
        I/O Size = 64KB
        Test Mode = Read write mix
        Start Time = 5/29/2013 2:57:00 PM
[Completion Info]
        Elapsed Time = 00:00:01
        Samples = 0
        I/O Halts = 0
        Avg Completion Time in Seconds = 0.000000
        Max Completion Time in Seconds = 0.000000
        Min Completion Time in Seconds = 0.000000
[I/O Operations]
        Total IOs = 0
        Avg IO/Sec = 0.00
        Max IO/Sec = 0.00
        Min IO/Sec = 0.00
[Megabytes]
        Total MB = 0
        Avg MB/Sec = 0.00
        Max MB/Sec = 0.00
        Min MB/Sec = 0.00
[Errors]
        Total Errors = 1
        Startup Errors = 1
```

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**

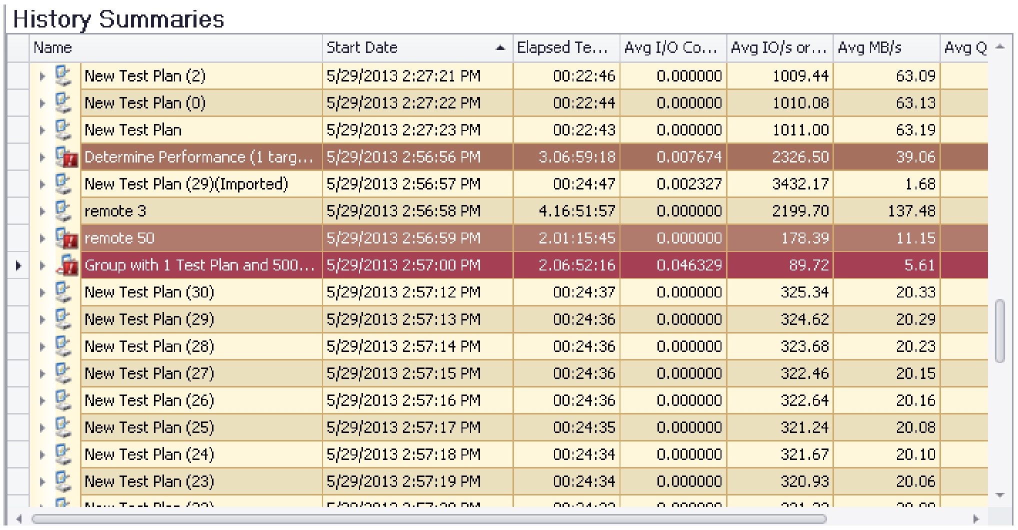
## History Summaries Pane

The **History Summaries** pane (see Figure 49) 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 74 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 49** History Summaries Pane

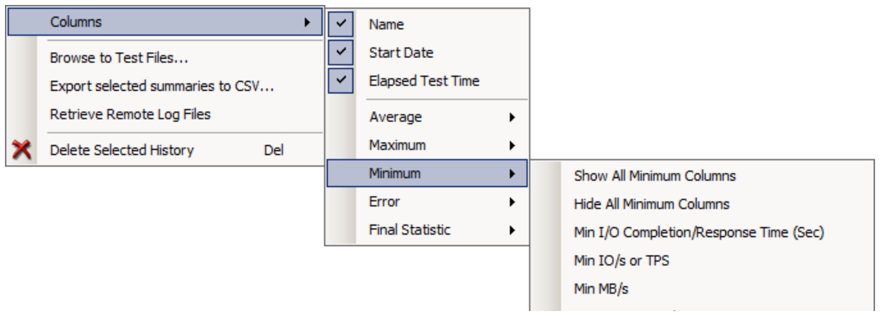


History Summaries

| Name | Start Date | Elapsed Te... | Avg I/O Co... | Avg IO/s or... | Avg MB/s | Avg Q |
|---|---|---|---|---|---|---|
| New Test Plan (2) | 5/29/2013 2:27:21 PM | 00:22:46 | 0.000000 | 1009.44 | 63.09 | |
| New Test Plan (0) | 5/29/2013 2:27:22 PM | 00:22:44 | 0.000000 | 1010.08 | 63.13 | |
| New Test Plan | 5/29/2013 2:27:23 PM | 00:22:43 | 0.000000 | 1011.00 | 63.19 | |
| Determine Performance (1 targ... | 5/29/2013 2:56:56 PM | 3.06:59:18 | 0.007674 | 2326.50 | 39.06 | |
| New Test Plan (29)(Imported) | 5/29/2013 2:56:57 PM | 00:24:47 | 0.002327 | 3432.17 | 1.68 | |
| remote 3 | 5/29/2013 2:56:58 PM | 4.16:51:57 | 0.000000 | 2199.70 | 137.48 | |
| remote 50 | 5/29/2013 2:56:59 PM | 2.01:15:45 | 0.000000 | 178.39 | 11.15 | |
| Group with 1 Test Plan and 500... | 5/29/2013 2:57:00 PM | 2.06:52:16 | 0.046329 | 89.72 | 5.61 | |
| New Test Plan (30) | 5/29/2013 2:57:12 PM | 00:24:37 | 0.000000 | 325.34 | 20.33 | |
| New Test Plan (29) | 5/29/2013 2:57:13 PM | 00:24:36 | 0.000000 | 324.62 | 20.29 | |
| New Test Plan (28) | 5/29/2013 2:57:14 PM | 00:24:36 | 0.000000 | 323.68 | 20.23 | |
| New Test Plan (27) | 5/29/2013 2:57:15 PM | 00:24:36 | 0.000000 | 322.46 | 20.15 | |
| New Test Plan (26) | 5/29/2013 2:57:16 PM | 00:24:36 | 0.000000 | 322.64 | 20.16 | |
| New Test Plan (25) | 5/29/2013 2:57:17 PM | 00:24:35 | 0.000000 | 321.24 | 20.08 | |
| New Test Plan (24) | 5/29/2013 2:57:18 PM | 00:24:34 | 0.000000 | 321.67 | 20.10 | |
| New Test Plan (23) | 5/29/2013 2:57:19 PM | 00:24:34 | 0.000000 | 320.93 | 20.06 | |

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 50.

**Figure 50** 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**

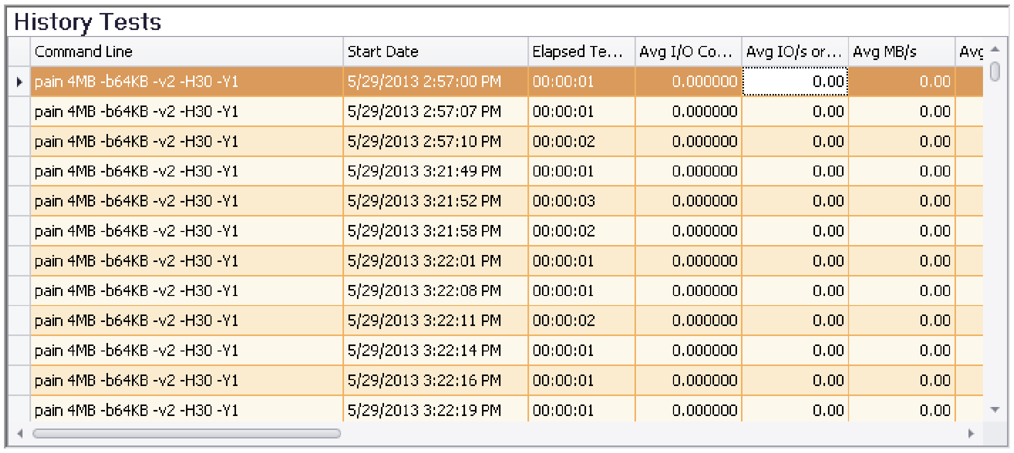
Right-clicking over a column displays all of the columns selections vertically without displaying the groups listed above. This may be used to save key strokes.

## History Tests Pane

The individual command line commands of the selected test are displayed in the **History Tests** pane (see Figure 51).

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 51** History Tests Pane



History Tests

| | Command Line | Start Date | Elapsed Te... | Avg I/O Co... | Avg IO/s or... | Avg MB/s |
|---|---|---|---|---|---|---|
| ▸ | 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 50) 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**

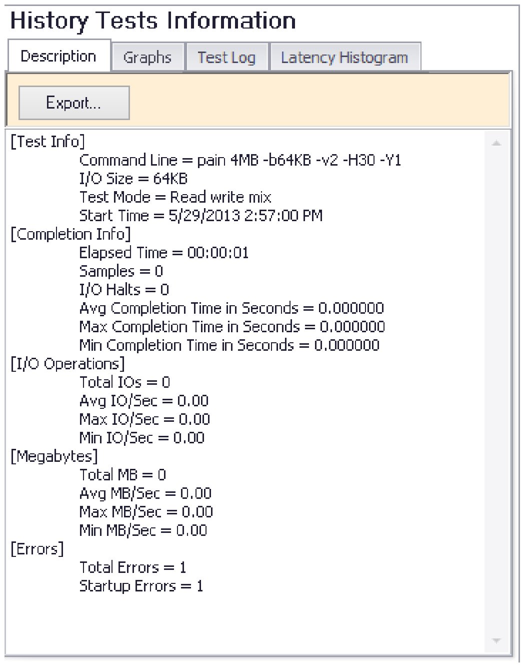
Right-clicking over a column displays all of the columns selections vertically without displaying the groups listed above. This may be used to save key strokes.

## 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 (see Figure 52), 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 52** History Information Pane (History Summaries Information Version)



```
[Test Info]
        Command Line = pain 4MB -b64KB -v2 -H30 -Y1
        I/O Size = 64KB
        Test Mode = Read write mix
        Start Time = 5/29/2013 2:57:00 PM
[Completion Info]
        Elapsed Time = 00:00:01
        Samples = 0
        I/O Halts = 0
        Avg Completion Time in Seconds = 0.000000
        Max Completion Time in Seconds = 0.000000
        Min Completion Time in Seconds = 0.000000
[I/O Operations]
        Total IOs = 0
        Avg IO/Sec = 0.00
        Max IO/Sec = 0.00
        Min IO/Sec = 0.00
[Megabytes]
        Total MB = 0
        Avg MB/Sec = 0.00
        Max MB/Sec = 0.00
        Min MB/Sec = 0.00
[Errors]
        Total Errors = 1
        Startup Errors = 1
```

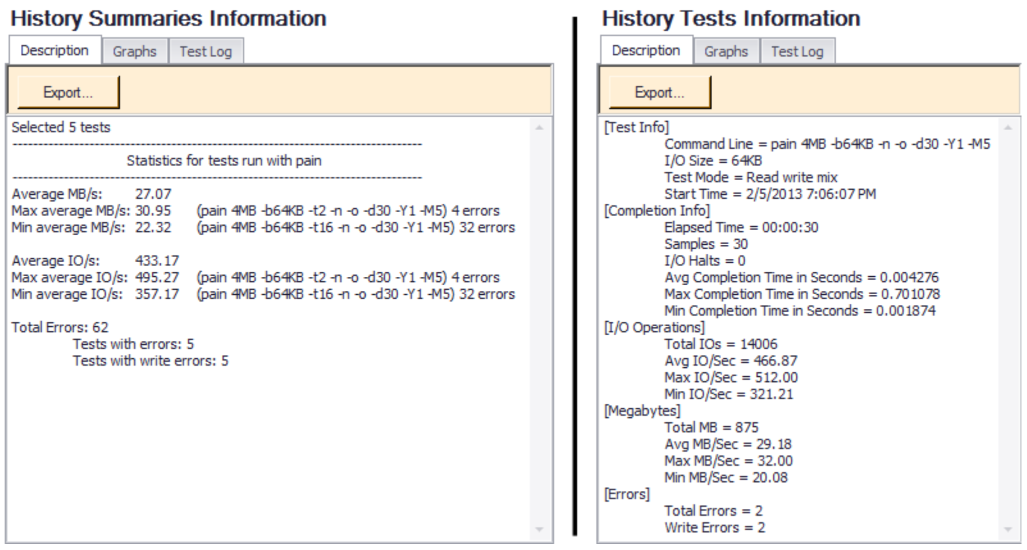
The History Information pane has four tabs:

- **Description** (see “Description Tab” on page 93)
- **Graphs** (see “Graphs Tab” on page 94)
- **Test Log** (see “Test Log Tab” on page 99)
- **Latency Histogram** (see “Latency Histogram Tab” on page 100)

## Description Tab

The **Description** tab provides a summary of the selected test from the **History Summaries** pane (shown on the left side of Figure 53) or the selected command line(s) from the **History Tests** pane (shown on the right side of Figure 53).

**Figure 53** Description Tab Examples



```
Selected 5 tests
-----------------------------------------------------------
              Statistics for tests run with pain
-----------------------------------------------------------
Average MB/s:     27.07
Max average MB/s: 30.95    (pain 4MB -b64KB -t2 -n -o -d30 -Y1 -M5) 4 errors
Min average MB/s: 22.32    (pain 4MB -b64KB -t16 -n -o -d30 -Y1 -M5) 32 errors

Average IO/s:     433.17
Max average IO/s: 495.27   (pain 4MB -b64KB -t2 -n -o -d30 -Y1 -M5) 4 errors
Min average IO/s: 357.17   (pain 4MB -b64KB -t16 -n -o -d30 -Y1 -M5) 32 errors

Total Errors: 62
        Tests with errors: 5
        Tests with write errors: 5
```

```
[Test Info]
        Command Line = pain 4MB -b64KB -n -o -d30 -Y1 -M5
        I/O Size = 64KB
        Test Mode = Read write mix
        Start Time = 2/5/2013 7:06:07 PM
[Completion Info]
        Elapsed Time = 00:00:30
        Samples = 30
        I/O Halts = 0
        Avg Completion Time in Seconds = 0.004276
        Max Completion Time in Seconds = 0.701078
        Min Completion Time in Seconds = 0.001874
[I/O Operations]
        Total IOs = 14006
        Avg IO/Sec = 466.87
        Max IO/Sec = 512.00
        Min IO/Sec = 321.21
[Megabytes]
        Total MB = 875
        Avg MB/Sec = 29.18
        Max MB/Sec = 32.00
        Min MB/Sec = 20.08
[Errors]
        Total Errors = 2
        Write Errors = 2
```

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 94 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.

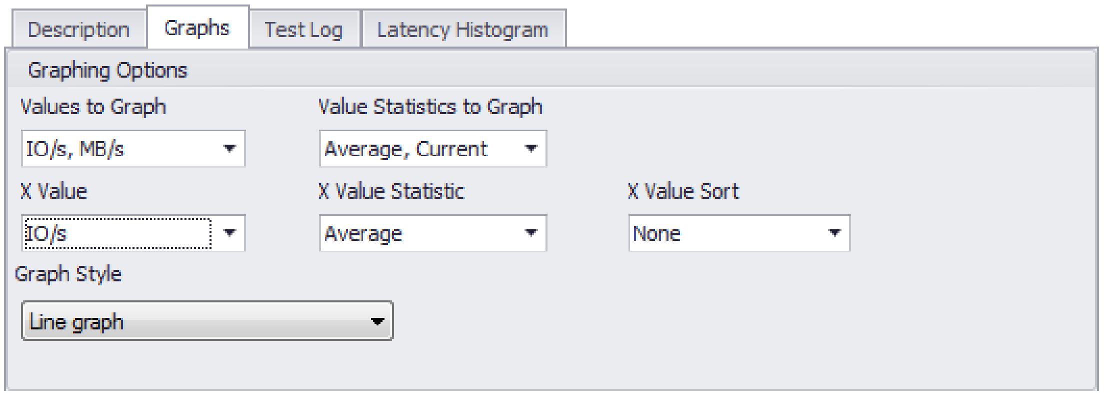
**NOTE**

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 (see Figure 54) provides several options that you may use to graph your test results. The options are **Values to Graph**, **Value Statistics to Graph**, **X Value**, **X Value Statistic**, **X Value Sort**, and **Graph Style**.

**Figure 54** Graphing Options Area



Values to Graph: IO/s, MB/s
Value Statistics to Graph: Average, Current
X Value: IO/s
X Value Statistic: Average
X Value Sort: None
Graph Style: Line graph

**Values to Graph** – selects 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.

**Value Statistics to Graph** – selects 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.

**X Value** – selects Default, IO/s, MB/s, I/O Completion/Response Time, or Queue Depth. Only one selection may be made. When a choice other than Default is selected, the **X Value Statistic** and **X Value Sort** options become active. The default selection is Default.

**X Value Statistic** – selects Average, Current, Maximum, or Minimum. Only one selection may be made. The default selection is Average.

**X Value Sort** – selects None, Ascending, or Descending. Only one selection may be made. The default selection is None. For this parameter, None means that the values are not sorted but are displayed in the same chronological order as the test was performed.

The **Graph Style** – selects 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.

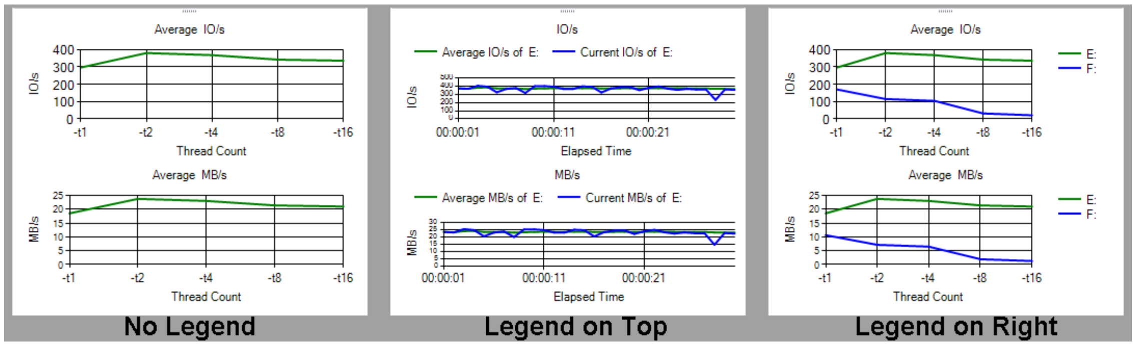
When you select a test plan in the **History Summaries** pane that uses both types of queuing (burst and continuous), their graphs will be grouped by their queuing type with burst queuing displayed on the **Burst** tab and continuous queuing displayed on the **Static** tab.

## 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 55.

- 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.

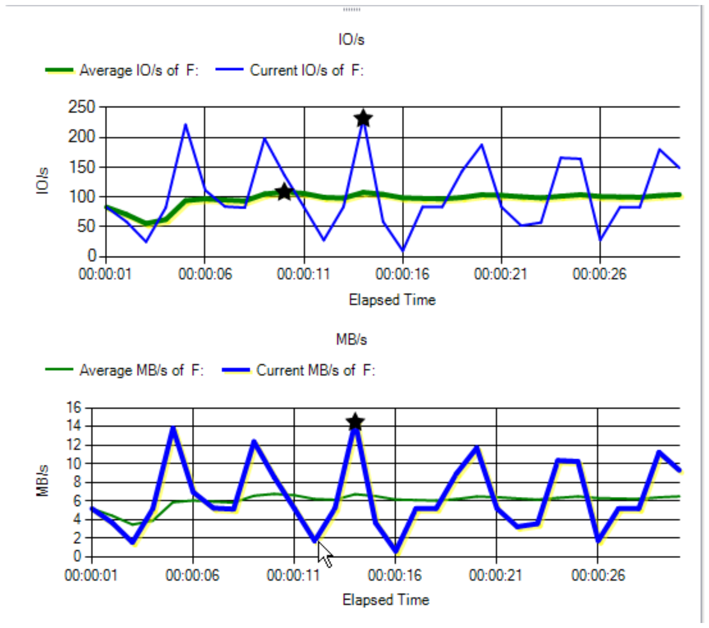
**Figure 55** Graphs Legends



## 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 star 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 56, the upper graph shows both plots displaying their highest value and the lower graph shows one plot after it was clicked.

**Figure 56** Displaying the Highest Value on a Plot

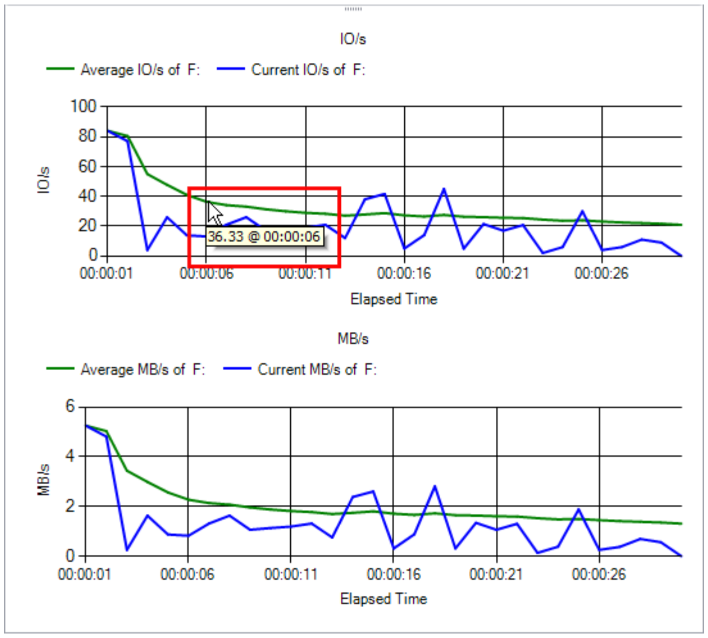


## Displaying Values on Line Graphs

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 57, hovering your cursor over a point on the plot displays the values of the graph's coordinates at that point.

**Figure 57** 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:

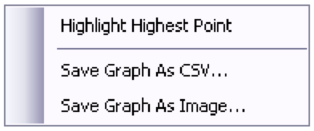
- mouse's scroll wheel
- keyboard's +/- keys or the "q"/"w" keys

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 58. 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.

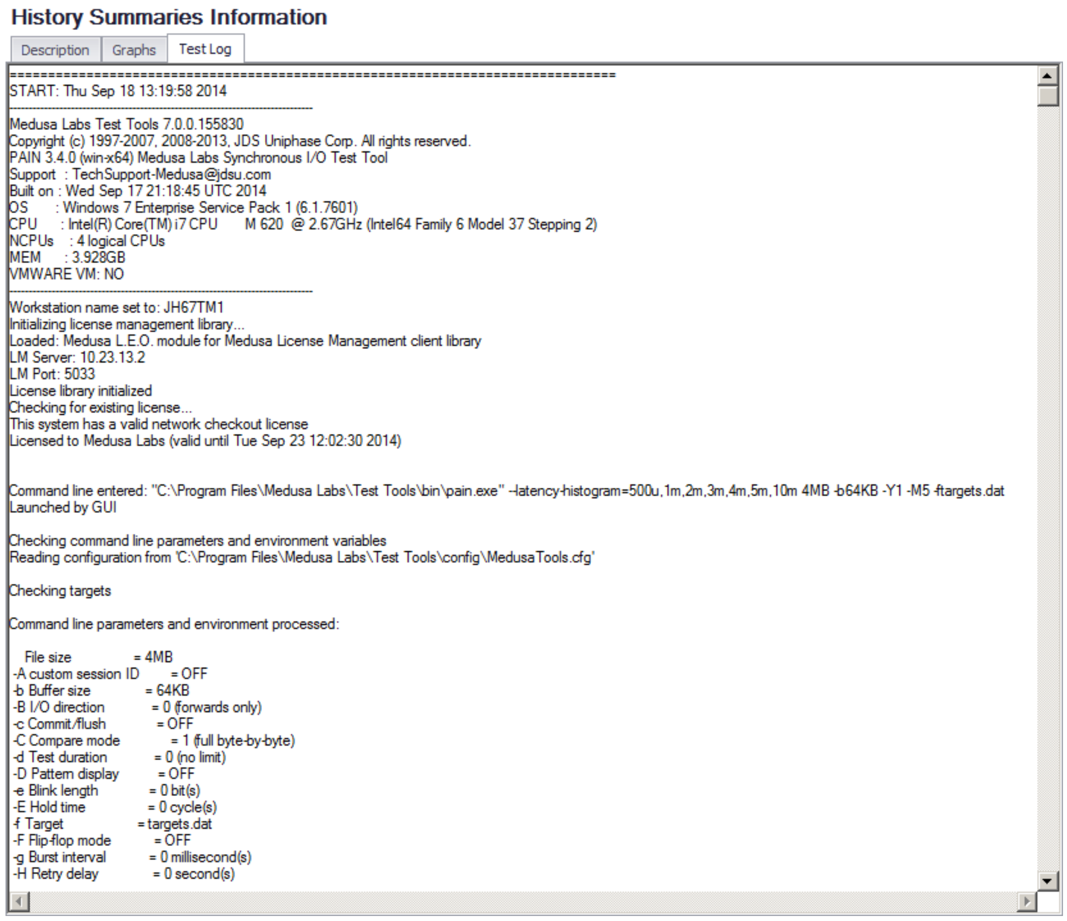
**Figure 58** Saving the Graph



## Test Log Tab

For the **Test Log** tab (see Figure 59), 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 59** Test Log Tab



```
==========================================================================
START: Thu Sep 18 13:19:58 2014
--------------------------------------------------------------------------
Medusa Labs Test Tools 7.0.0.155830
Copyright (c) 1997-2007, 2008-2013, 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 17 21:18:45 UTC 2014
OS       : Windows 7 Enterprise Service Pack 1 (6.1.7601)
CPU      : Intel(R) Core(TM) i7 CPU       M 620  @ 2.67GHz (Intel64 Family 6 Model 37 Stepping 2)
NCPUs    : 4 logical CPUs
MEM      : 3.928GB
VMWARE VM: NO
--------------------------------------------------------------------------
Workstation name set to: JH67TM1
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 Tue Sep 23 12:02:30 2014)


Command line entered: "C:\Program Files\Medusa Labs\Test Tools\bin\pain.exe" --latency-histogram=500u,1m,2m,3m,4m,5m,10m 4MB -b64KB -Y1 -M5 -ftargets.dat
Launched by GUI

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                = targets.dat
-F Flip-flop mode        = OFF
-g Burst interval        = 0 millisecond(s)
-H Retry delay           = 0 second(s)
```

## Latency Histogram Tab

Latency histogram collects latency histogram per target. The collection bins are specified when using the "Custom Configuration Editor" on page 100 and the "Performance Configuration Editor" on page 132. 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 (see Figure 60).

**IMPORTANT**

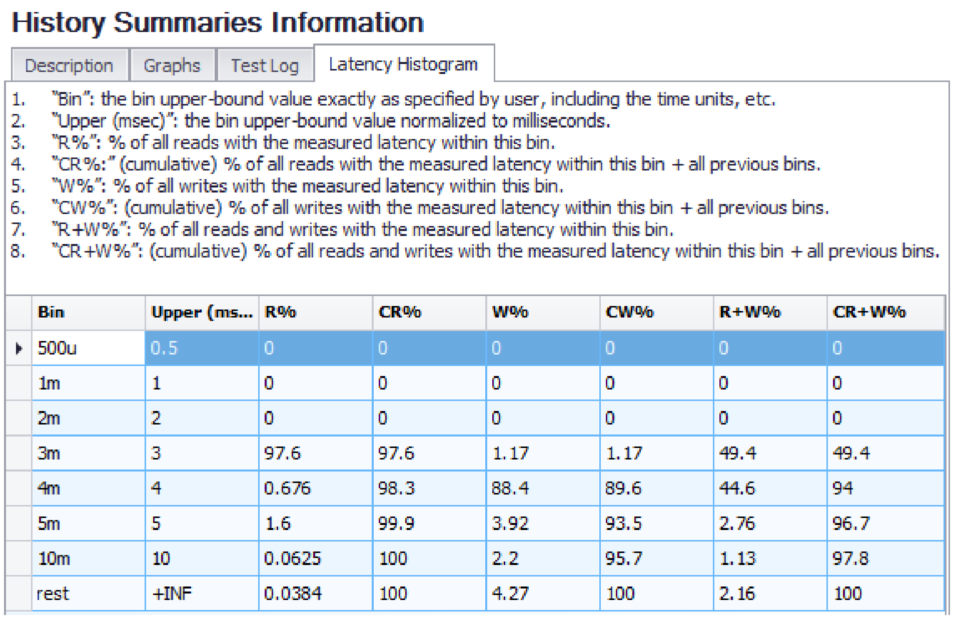
The Latency Histogram table (with its data) is only displayed when the drive (of a configuration) utilizing Latency Histogram is selected in the **History Summaries** pane. If no drive has been selected, the **Latency Histogram** tab advises you to select a drive.

**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 60** Latency Histogram Tab



1. "Bin": the bin upper-bound value exactly as specified by user, including the time units, etc.
2. "Upper (msec)": the bin upper-bound value normalized to milliseconds.
3. "R%": % of all reads with the measured latency within this bin.
4. "CR%:" (cumulative) % of all reads with the measured latency within this bin + all previous bins.
5. "W%": % of all writes with the measured latency within this bin.
6. "CW%": (cumulative) % of all writes with the measured latency within this bin + all previous bins.
7. "R+W%": % of all reads and writes with the measured latency within this bin.
8. "CR+W%": (cumulative) % of all reads and writes with the measured latency within this bin + all previous bins.

| 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 |

3

# Using the Configuration Editors

This chapter describes how to set up the Medusa Labs Test Tools Suite configuration using the configuration editors. The topics discussed in this chapter are as follows:

## Using the GUI Configuration Editors

This chapter provides detailed information for editing configurations. Configurations will normally be edited when creating new configurations in the “Configurations Area” on page 63 or when editing existing configurations with the “Configuration Editor” on page 76 of the Test Plans Editor pane. Each of these areas has the **New Configuration** button described below. Each of the configuration editors are described in this chapter.

**NOTE**

Configuration Editors are accessible using a variety of methods in the Configurations area and the Test Plans area. To access configurations from the:

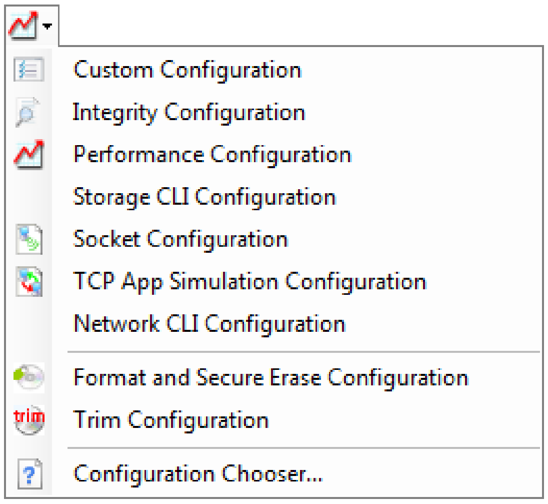
– Configurations area, refer to “Configurations Area” on page 63

– Test Plans area, refer to “New Test Plan Button” on page 66

### New Configuration Button

This button opens the following drop-down list of configurations. Selecting one of the 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.

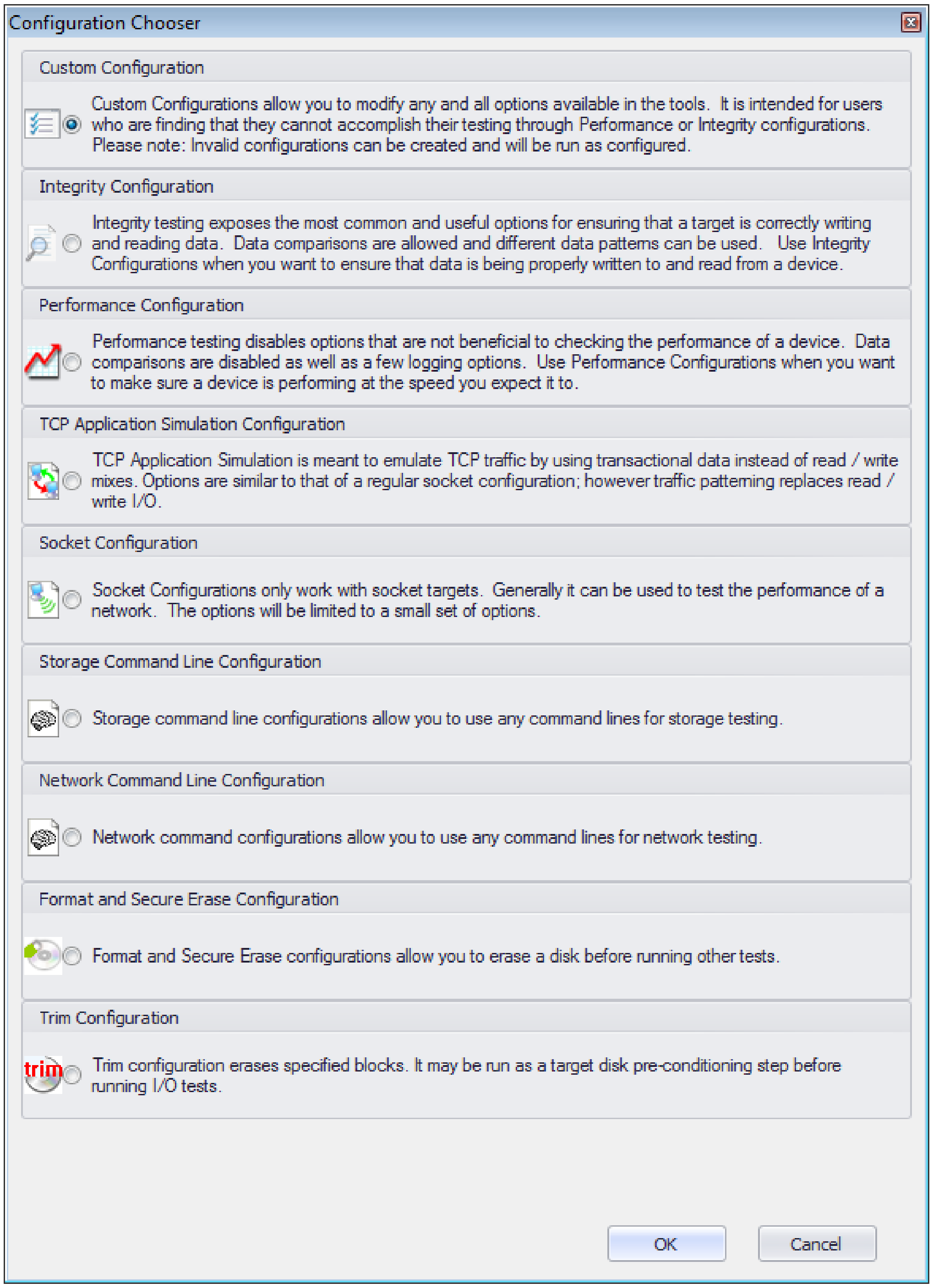
**Figure 61** Create New Configuration Button


Custom Configuration
Integrity Configuration
Performance Configuration
Storage CLI Configuration
Socket Configuration
TCP App Simulation Configuration
Network CLI Configuration
Format and Secure Erase Configuration
Trim Configuration
Configuration Chooser...

## Configuration Chooser Window

The **Configuration Chooser** window (see Figure 62) allows you to select the configuration for the test that you want to run.

**Figure 62** Configuration Chooser Window



Configuration Chooser

Custom Configuration

[✓] Custom Configurations allow you to modify any and all options available in the tools. It is intended for users who are finding that they cannot accomplish their testing through Performance or Integrity configurations. Please note: Invalid configurations can be created and will be run as configured.

Integrity Configuration

Integrity 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 Configurations when you want to ensure that data is being properly written to and read from a device.

Performance Configuration

Performance 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 Configurations when you want to make sure a device is performing at the speed you expect it to.

TCP Application Simulation Configuration

TCP Application Simulation 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.

Socket Configuration

Socket Configurations only work with socket targets. Generally it can be used to test the performance of a network. The options will be limited to a small set of options.

Storage Command Line Configuration

Storage command line configurations allow you to use any command lines for storage testing.

Network Command Line Configuration

Network command configurations allow you to use any command lines for network testing.

Format and Secure Erase Configuration

Format and Secure Erase configurations allow you to erase a disk before running other tests.

Trim Configuration

Trim configuration erases specified blocks. It may be run as a target disk pre-conditioning step before running I/O tests.

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 106.
- **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 125.
- **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 138.
- **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 146.
- **Socket Configuration**
  Socket Configuration only works with socket targets. Generally this configuration can be used to test the performance of a network. For more information on how to edit socket configuration settings, see "Socket Configuration Editor" on page 147.
- **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 160.
- **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 171.
- **Format and Secure Erase**
  Format and Secure Erase Configuration allows you to erase a disk before running other tests. For information on using the Format and Secure Erase configuration editor, see "Format and Secure Erase Configuration Editor" on page 172.

- **Trim**
  Trim Configuration allows you to erase unused blocks to pre-condition a target disk before running other tests. For information on using the Trim configuration editor, see "Trim Configuration Editor" on page 176.

## Configuration Editors

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

- "Custom Configuration Editor" on page 106
- "Integrity Configuration Editor" on page 125
- "Performance Configuration Editor" on page 138
- "Storage CLI Configuration Editor" on page 146
- "Socket Configuration Editor" on page 147
- "TCP App Simulation Configuration Editor" on page 160
- "Network CLI Configuration Editor" on page 171
- "Format and Secure Erase Configuration Editor" on page 172
- "Trim Configuration Editor" on page 176

### 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 -t64pain -t1`<br>`pain -t2`<br>`pain -t3`<br>`pain -t4`<br>…<br>`pain -t63` | `pain -t1`<br>`pain -t2`<br>`pain -t4`<br>`pain -t8`<br>`pain -t16`<br>`pain -t32`<br>`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 GUI Configuration Editors” on page 102.

**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 eight tabs for specifying testing parameters:

- “General Tab” on page 106
- “Journal Tab” on page 108
- “I/O Payload Tab” on page 110
- “I/O Behavior Tab” on page 115
- “Advanced I/O Tab” on page 118
- “Patterns Tab” on page 119
- “Comments Tab” on page 123
- “Command Lines Tab” on page 123

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 between starting the test and the first sample collection. 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.

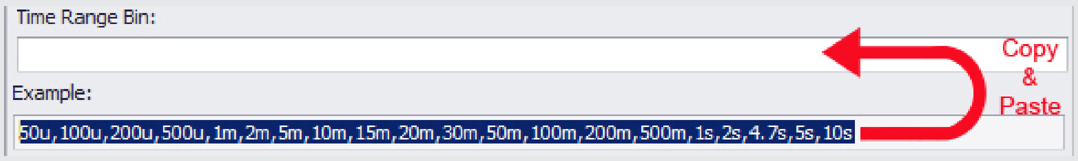
The collection bins are specified by entering the upper bound of each bin as a comma-separated list in the **Time Range Bin** text box. The list is sorted by the magnitude of the upper bound values, and the range of each bin is constructed such that the upper bound is as specified and the lower bound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


Time Range Bin:
Example:
50u,100u,200u,500u,1m,2m,5m,10m,15m,20m,30m,50m,100m,200m,500m,1s,2s,4.7s,5s,10s

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

The "Latency Histogram Tab" on page 93 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%.

## Journal Tab

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

**Setting** –

**Enable** – enables (or disables) either the Perform (Journal) or the Verify options by selecting (or deselecting) this check box.

**Perform** – records a log file of recent write operation characteristics (buffer size, thread count, queue depth, file size, pattern used, etc.). Then, when a power loss to the initiator or the target device is simulated during testing, the log file is saved. The log file preserves the status of the last several write operations. A path and directory must be specified as a location to save the .log file using the **Directory** text box (or by using the **Browse...** button.) The specified directory must already exist.

**NOTE**

The GUI does not browse remote host file systems. The **Browse...** button only works for tests on the local system.

You can specify a raw disk as the journal directory using the **Directory** text box. For example:

- on Windows you would specify: `\\.\PhysicalDrive1`
- on Linux you would specify: `/dev/sdb`

Journal data is written starting at 1MB offset of the specified disks. Using a raw disk can reduce the journal data update latency and better ensure the journal data integrity.

**Verify** – when the power is restored, this selection retrieves the saved log file and verifies the last known completed write operations prior to the power loss.

**NOTE**
When Verify is selected, only the Journal, I/O Payload, Comments, and Command Lines tabs are displayed. The General, I/O Behavior, Advanced I/O, and Patterns tabs are hidden because they are not applicable when verification is being performed.

If a path and directory was identified in the initial save portion of the process, the same path and directory must be identified during the verify process. Specify the path and directory of the saved .log file using the **Directory** text box (or the **Browse...** button). The journal file is created in the working directory that the tools run from by default. A best practice is to always specify the journal location so that the log can be found during a verify operation.

**NOTE**
When you perform journaling in pain, verification must also be done in pain. Likewise, when you perform journaling in maim, verification must also be done in maim.

When the journal verification is run, the following summary information is output to the screen and the log file at the end of the test:

```
JOURNAL VERIFICATION SUMMARY
    TOTAL (both queued and confirmed writes):
        240
    CONFIRMED (confirmed writes):
        227
    VERIFIED (queued or confirmed writes that passed):
        239
    FAILED (confirmed writes with data corruption):
        0
    SKIPPED (skipped verification due to overwrite by newer write(s)):
        0
    UNDEFINED (queued writes that failed verification):
        1
```

In addition, the log file also lists the details for each recorded write operation that was verified. Here are examples of three records:

```
JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:0
    JOURNAL: OFFSET    : 4377804800 (0x0000000104F00000)
    JOURNAL: LBA       : 8550400 (0x0000000000827800)
    JOURNAL: SIZE      : 64KB
    JOURNAL: LOOP      : 0
    JOURNAL: TIMESTAMP : 0x51AC51A7E2B9E010
    JOURNAL: WRITE     : QUEUED
    JOURNAL: VERIFY    : OK

JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:0
    JOURNAL: OFFSET    : 4377673728 (0x0000000104EE0000)
    JOURNAL: LBA       : 8550144 (0x0000000000827700)
    JOURNAL: SIZE      : 64KB
    JOURNAL: LOOP      : 0
```

```
        JOURNAL: TIMESTAMP : 0x51AC51A7E2B9E00E
        JOURNAL: WRITE     : CONFIRMED
        JOURNAL: VERIFY    : OK

    JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:15
        JOURNAL: OFFSET    : 11024896 (0x0000000000A83A00)
        JOURNAL: LBA       : 21533 (0x000000000000541D)
        JOURNAL: SIZE      : 64KB
        JOURNAL: LOOP      : 0
        JOURNAL: TIMESTAMP : 0x51AC55D16454B001
        JOURNAL: WRITE     : CONFIRMED
        JOURNAL: VERIFY    : SKIP (OVERWRITE @ 0x51AC55D16BF5D000)
```

In these examples:

"OFFSET:", "LBA:", and "SIZE:" detail the I/O location and transfer size.

"LOOP:" is the sequential I/O loop count during which the recorded write occurred.

"TIMESTAMP:" is a concatenation of 52-bit microsecond time stamp and 12-bit sequence number to uniquely identify a recorded write. This number is unique within each I/O context ("CTX:").

"WRITE:" status can be "QUEUED" or "CONFIRMED". Before each write, the record for the write is set to "QUEUED" state and committed to the journal. When the write operation completes, the record for the write is set to "CONFIRMED" state and committed to the journal.

"VERIFY:" status is set during journal verification as each recorded data is read back and a data integrity check is performed on it.

The status indicators are set based on the following definitions:

**`OK`** if "CONFIRMED" or "QUEUED" write passes data integrity check.

**`FAIL`** if a "CONFIRMED" write fails data integrity check. This also raises the regular "Data corruption" error.

**`SKIP`** if the pain/maim detects that the I/O location was overwritten by a later write operation, and data verification is skipped in order to avoid falsely raising a data corruption error. In addition, the "TIME-STAMP" corresponding to over-writing I/O is given next to the status.

**`UNDEFINED`** if a "QUEUED" write (i.e. queued but not confirmed write operation) fails data integrity check. Because the write was not confirmed, this is not flagged as a data corruption error.

## 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 6** Performance Mode Settings

| GUI Setting | Command Line Setting |
|---|---|
| **I/O Marking and Signing** is set to **No I/O Markings**<br>No I/O signatures are applied to each sector of every write. | `-u` (page 233) |
| **Data Compare Mode** is set to **Disable Data Comparisons**<br>Data comparisons are turned off. | `-n` (page 233) |
| **I/O Behavior** is set to **Keep File Handles Open Between I/Os**<br>Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. | `-o` (page 218) |
| **Use Pattern Reversals** check box is not selected (cleared)<br>Leaves the data patterns reversal after each FOP (forward, then backward) turned off. | `-N` (page 228) |

**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, if use of the SCSI passthough mode is desired. The options are listed below:

| | |
|---|---|
| **SCSI Passthrough Off** | Not using direct SCSI command. |
| **READ/WRITE 10** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 10 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **SCSI UNMAP / ATA TRIM** | Supports the TRIM command. A more complete description is provided below. |
| **READ/WRITE 32** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |

| | |
|---|---|
| **READ/WRITE 32 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |

The SCSI UNMAP / ATA TRIM option turns on SCSI UNMAP (or ATA TRIM) as a write operation for normal I/O engine tests. Reads are done using the normal operating system functions, while writes are replaced with UNMAP or TRIM to the requested offset using buffer size. The assumption is that most devices will return buffers filled with “0’s” for reads after TRIM (without any write in-between).

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

**Testing Threads** – Set the threads for testing. With Pain, each thread executes a single I/O at a time, with each thread starting at a different base offset. With Maim, tests will issue multiple IO requests per thread, equal to the defined queue depth. 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 105 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 105 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 105 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 I/O size. For asynchronous tests (maim), the file size is the I/O 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 (reading back the same areas that were written).

**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. Using this option will allow for data comparison during the remaining read-only portion of the test.

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

**Write Only** – Select this option to perform 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.
- If any Read/Write mix is used (including 50%-50%), the reads will have no correlation to the writes (e.g. it is not reading back the same data that was written).

**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.

The **Enable Random Offset Alignment** check box enables the random offset alignment size. You can set the random offset alignment size to ensure that random I/Os are issued on boundaries of the indicated size. This setting is available when the **% Random** value is greater than 0.

You can set the random offset alignment size by entering the value or clicking the numeric spinner. The minimum value is equal to the size of the sector. The size unit can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, **EB**, or **Units** from the drop-down menu.

**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 K "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.

**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 ASCII representation 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** – (displayed for Asynchronous (Maim) only) 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 saves the contents of the read and write buffers for the transaction. 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. It also saves the contents of the read and write buffers for the transaction and are also extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer. 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. See “-! (or -#) Enable Analyzer trigger writes” on page 240 for additional information.

**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 drop-down 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**.

The **Enable Random Offset Alignment** check box enables the random offset alignment size. You can set the random offset alignment size to ensure that random I/Os are issued on boundaries of the indicated size. This setting is available when the **% Random** value is greater than 0.

You can set the random offset alignment size by entering the value or clicking the numeric spinner. The minimum value is equal to the size of the sector. The size unit can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, **EB**, or **Units** from the drop-down menu.

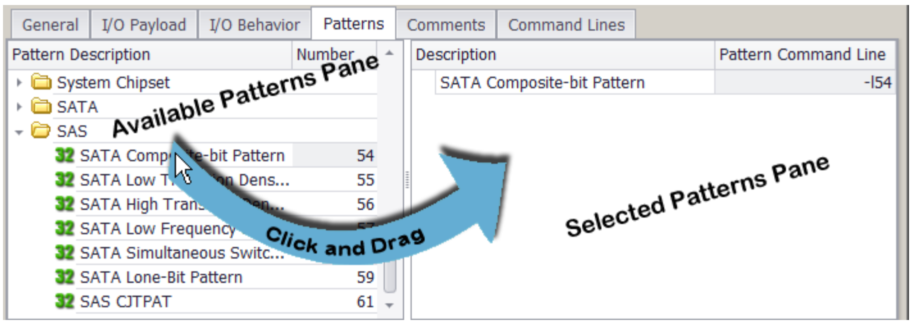
## Patterns Tab

The **Patterns** tab (see Figure 63) 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 63** 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 **Specify Phase Cycle Length** is selected, enter the number of cycle units to run before doing the phase shift. Change the number of units 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 do not want to use with the default. All data patterns that use the **Data Pattern Specification** field default to an all-zero value in the GUI (the length depends on the size of the repeating value data pattern, but it always defaults to all zeros).

The **Data Pattern Specification** setting corresponds to the CLI data pattern option "`-y`<hex value>", which does default to the thread number if it is not explicitly set in a CLI command (but only in the CLI). 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 7. The opposing bits are shaded in the table so the walking effect can be seen easily.

**Table 7** 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 the specified number of cycles 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 the blink length by entering a value in the text box or using the numeric spinner.

The following settings applies when the **Compression & Dedup** pattern is selected:

**Random Seed**

**Seed** – Select the check box to specify a 32-bit random number seed value. Enter a random seed value by entering a value in the text box or using the numeric spinner.

**Compression and Deduplication Settings**

**Compression** / **Deduplication** / **Both** radio buttons – Select the option that your testing requires. Select **Compression** for compression testing only; select **Deduplication** for deduplication testing only; or select **Both** for both of the previous options. This selection affects which of the following compression/deduplication options are active.

**Entropy Strength** – Specifies how compressible the payload is.
0 (no entropy, most compressible) to 100 (most entropy, least compressible). It basically defines the percentage of original data that is written after compression is applied. Change the entropy strength by entering a value in the text box or

using the numeric spinner.
For example, when the value is set to 25, this results in data output that is about 25% of the original data size after compression with typical data compression algorithms. The default value is 100.

**Dedup %** – Specifies the percentage of written data that can be deduplicated by the target device. The percentage of duplicated data blocks can be specified from 0 to 100 percent. Change the dedup percentage by entering a value in the text box or using the numeric spinner. For example, using a value of 30 results in about 70% of generated data being unique and about 30% being filled from a pool of duplicate blocks that can repeated (thus be deduplicated by the target device.) The default value is 0.

**Duplicate Block** – Sets the number of unique blocks in the duplicate block pool to draw from. Change the number of unique blocks by entering a value in the text box or using the numeric spinner. The block correlates with whatever dedup block size that is used by the target's dedup engine and the default block size is set to 8KB. From the second time a block from this pool is written out, it can be deduplicated. The default value is 1.

**Dedup Block Size** – Specifies the dedup block size. It should be set to whatever the value the target storage device's deduplication system uses. Change the block size by entering a value in the text box or using the numeric spinner. The "units" list allows you to select either kilobytes (KB) or megabytes (MB). The default for MLTT is 8 kilobytes. The maximum allowable dedup block size is 1032MB.

### 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 generated 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 GUI Configuration Editors" on page 102.

**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 Tab" on page 125
- "I/O Payload Tab" on page 126
- "I/O Behavior Tab" on page 130
- "Patterns Tab" on page 133
- "Comments Tab" on page 137
- "Command Lines Tab" on page 137

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 between starting the test and the first sample collection. 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 8** Performance Mode Settings

| GUI Setting | Command Line Setting |
|---|---|
| **I/O Marking and Signing** is set to **No I/O Markings**<br>No I/O signatures are applied to each sector of every write. | `-u` (page 233) |
| **Data Compare Mode** is set to **Disable Data Comparisons**<br>Data comparisons are turned off. | `-n` (page 233) |
| **I/O Behavior** is set to **Keep File Handles Open Between I/Os**<br>Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. | `-o` (page 218) |
| **Use Pattern Reversals** check box is not selected (cleared)<br>Leaves the data patterns reversal after each FOP (forward, then backward) turned off. | `-N` (page 228) |

**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, if use of the SCSI passthough mode is desired. The options are listed below:

| | |
|---|---|
| **SCSI Passthrough Off** | Not using direct SCSI command. |
| **READ/WRITE 10** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 10 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |

| | |
|---|---|
| **READ/WRITE 16 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **SCSI UNMAP / ATA TRIM** | Supports the TRIM command. A more complete description is provided below. |
| **READ/WRITE 32** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |
| **READ/WRITE 32 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |

The SCSI UNMAP / ATA TRIM option turns on SCSI UNMAP (or ATA TRIM) as a write operation for normal I/O engine tests. Reads are done using the normal operating system functions, while writes are replaced with UNMAP or TRIM to the requested offset using buffer size. The assumption is that most devices will return buffers filled with “0’s” for reads after TRIM (without any write in-between).

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

**Testing Threads** – Set the threads for testing. With Pain, each thread executes a single I/O at a time, with each thread starting at a different base offset. With Maim, tests will issue multiple IO requests per thread, equal to the defined queue depth. 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 105 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 105 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 105 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 I/O size. For asynchronous tests (maim), the file size is the I/O 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.

The **Enable Random Offset Alignment** check box enables the random offset alignment size. You can set the random offset alignment size to ensure that random I/Os are issued on boundaries of the indicated size. This setting is available when the **% Random** value is greater than 0.

You can set the random offset alignment size by entering the value or clicking the numeric spinner. The minimum value is equal to the size of the sector. The size unit can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, **EB**, or **Units** from the drop-down menu.

**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 K "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.
>
> **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 ASCII representation 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** – (displayed for Asynchronous (Maim) only) 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 saves the contents of the read and write buffers for the transaction. 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. It also saves the contents of the read and write buffers for the transaction and are also extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer. 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. See ““-! (or -#) Enable Analyzer trigger writes” on page 240 for additional information.

**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**.

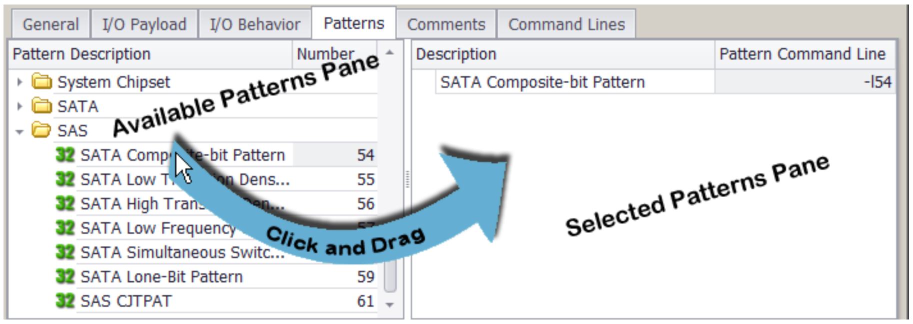
## Patterns Tab

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 64** Available Patterns and Selected Patterns



| Pattern Description | Number |
|---|---|
| System Chipset | |
| SATA | |
| SAS | |
| SATA Composite-bit Pattern | 54 |
| SATA Low Tr... Dens... | 55 |
| SATA High Trans... | 56 |
| SATA Low Frequency | |
| SATA Simultaneous Switc... | |
| SATA Lone-Bit Pattern | 59 |
| SAS CJTPAT | 61 |

| Description | Pattern Command Line |
|---|---|
| SATA Composite-bit Pattern | -I54 |

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 **Specify Phase Cycle Length** is selected, enter the number of cycle units to run before doing the phase shift. Change the number of units 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. All data patterns that use the **Data Pattern Specification** field default to an all-zero value in the GUI (the length depends on the size of the repeating value data pattern, but it always defaults to all zeros).

The Data Pattern Specification setting corresponds to the CLI data pattern option "-y<hex value>", which does default to the thread number if it is not explicitly set in a CLI command (but only in the CLI). 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 9. The opposing bits are shaded in the table so the walking effect can be seen easily.

**Table 9** 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 the specified number of cycles 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 the blink length by entering a value in the text box or using the numeric spinner.

The following settings applies when the **Compression & Dedup** pattern is selected:

**Random Seed**

**Seed** – Select the check box to specify a 32-bit random number seed value. Enter a random seed value by entering a value in the text box or using the numeric spinner.

**Compression and Deduplication Settings**

**Compression** / **Deduplication** / **Both** radio buttons – Select the option that your testing requires. Select **Compression** for compression testing only; select **Deduplication** for deduplication testing only; or select **Both** for both of the previous options. This selection affects which of the following compression/deduplication options are active.

**Entropy Strength** – Specifies how compressible the payload is.
0 (no entropy, most compressible) to 100 (most entropy, least compressible). It basically defines the percentage of original data that is written after compression is applied. Change the entropy strength by entering a value in the text box or using the numeric spinner. For example, when the value is set to 25, this results in data output that is about 25% of the original data size after compression with typical data compression algorithms. The default value is 100.

**Dedup %** – Specifies the percentage of written data that can be deduplicated by the target device. The percentage of duplicated data blocks can be specified from 0 to 100 percent. Change the dedup percentage by entering a value in the text box or using the numeric spinner. For example, using a value of 30 results in about 70% of generated data being unique and about 30% being filled from a pool of duplicate blocks that can repeated (thus be deduplicated by the target device.) The default value is 0.

**Duplicate Block** – Sets the number of unique blocks in the duplicate block pool to draw from. Change the number of unique blocks by entering a value in the text box or using the numeric spinner. The block correlates with whatever dedup block size that is used by the target's dedup engine and the default block size is set to 8KB. From the second time a block from this pool is written out, it can be deduplicated. The default value is 1.

**Dedup Block Size** – Specifies the dedup block size. It should be set to whatever the value the target storage device’s deduplication system uses. Change the block size by entering a value in the text box or using the numeric spinner. The “units” list allows you to select either kilobytes (KB) or megabytes (MB). The default for MLTT is 8 kilobytes. The maximum allowable dedup block size is 1032MB.

## 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 generated 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 GUI Configuration Editors" on page 102.

**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 Tab" on page 138
– "I/O Payload Tab" on page 140
– "Comments Tab" on page 145
– "Command Lines Tab" on page 145

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 between starting the test and the first sample collection. 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.

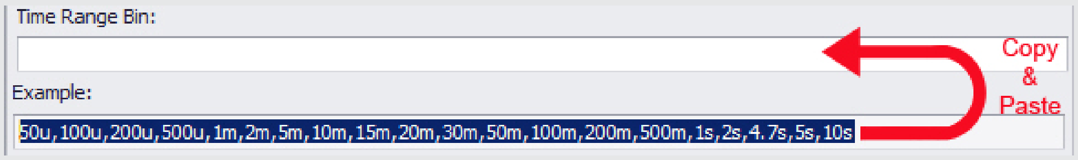
Upper bounds may be specified as a floating point value (e.g. "0.5" or "4.5").
The time unit suffix may be used:

'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.



50u,100u,200u,500u,1m,2m,5m,10m,15m,20m,30m,50m,100m,200m,500m,1s,2s,4.7s,5s,10s

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

The Latency Histogram tab (see page 93) 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 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 10** Performance Mode Settings

| GUI Setting | Command Line Setting |
|---|---|
| **I/O Marking and Signing** is set to **No I/O Markings**<br>No I/O signatures are applied to each sector of every write. | `-u` (page 233) |
| **Data Compare Mode** is set to **Disable Data Comparisons**<br>Data comparisons are turned off. | `-n` (page 233) |
| **I/O Behavior** is set to **Keep File Handles Open Between I/Os**<br>Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. | `-o` (page 218) |
| **Use Pattern Reversals** check box is not selected (cleared)<br>Leaves the data patterns reversal after each FOP (forward, then backward) turned off. | `-N` (page 228) |

**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, if use of the SCSI passthough mode is desired. The options are listed below:

| | |
|---|---|
| **SCSI Passthrough Off** | Not using direct SCSI command. |
| **READ/WRITE 10** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 10 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **SCSI UNMAP / ATA TRIM** | Supports the TRIM command. A more complete description is provided below. |
| **READ/WRITE 32** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |
| **READ/WRITE 32 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |

The SCSI UNMAP / ATA TRIM option turns on SCSI UNMAP (or ATA TRIM) as a write operation for normal I/O engine tests. Reads are done using the normal operating system functions, while writes are replaced with UNMAP or TRIM to the requested offset using buffer size. The assumption is that most devices will return buffers filled with “0’s” for reads after TRIM (without any write in-between).

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

**Testing Threads** – Set the threads for testing. With Pain, each thread executes a single I/O at a time, with each thread starting at a different base offset. With Maim, tests will issue multiple IO requests per thread, equal to the defined queue depth. 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 105 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 105 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 105 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 I/O size. For asynchronous tests (maim), the file size is the I/O 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 (reading back the same areas that were written).

**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. Using this option will allow for data comparison during the remaining read-only portion of the test.

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

**Write Only** – Select this option to perform 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.
– If any Read/Write mix is used (including 50%-50%), the reads will have no correlation to the writes (e.g. it is not reading back the same data that was written).

**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.

The **Enable Random Offset Alignment** check box enables the random offset alignment size. You can set the random offset alignment size to ensure that random I/Os are issued on boundaries of the indicated size. This setting is available when the **% Random** value is greater than 0.

You can set the random offset alignment size by entering the value or clicking the numeric spinner. The minimum value is equal to the size of the sector. The size unit can be set to **Bytes**, **KB**, **MB**, **GB**, **TB**, **PB**, **EB**, or **Units** from the drop-down menu.

**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 K "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 generated 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 allows pain and maim command lines to be run from the MLTT GUI. This method also allows the use of seldom used MLTT Command Line options that have not been exposed in the MLTT GUI configuration templates.

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 GUI Configuration Editors” on page 102.

**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 Lines
- 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.

The Storage CLI Configuration editor only accepts pain or maim commands. For sock commands, refer to “Network CLI Configuration Editor” on page 171.

### 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 GUI Configuration Editors" on page 102.

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

The editor has seven tabs for specifying testing parameters:

– "General Tab" on page 147)
– "I/O Payload Tab" on page 148)
– "I/O Behavior Tab" on page 151)
– "Advanced I/O Tab" on page 154)
– "Patterns Tab" on page 154
– "Comments Tab" on page 158
– "Command Lines Tab" on page 158

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 between starting the test and the first sample collection. 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 11** Performance Mode Settings

| GUI Setting | Command Line Setting |
|---|---|
| **I/O Marking and Signing** is set to **No I/O Markings**<br>No I/O signatures are applied to each sector of every write. | `-u` (page 233) |
| **Data Compare Mode** is set to **Disable Data Comparisons**<br>Data comparisons are turned off. | `-n` (page 233) |
| **I/O Behavior** is set to **Keep File Handles Open Between I/Os**<br>Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. | `-o` (page 218) |
| **Use Pattern Reversals** check box is not selected (cleared)<br>Leaves the data patterns reversal after each FOP (forward, then backward) turned off. | `-N` (page 228) |

**Testing Threads** – Set the threads for testing. Each thread executes a single I/O at a time. 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 105 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 105 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, the **Units** option refers to a fixed 512 bytes.

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 105 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.
>
> **Specify Testing Area** – Select this button and specify the size of data portions to be counted for the iteration counter.

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

**NOTE**

In the case of sockets, the term "write" is used to refer to the TCP or UDP stream from the initiator system(s) to the target system(s), and the term "read" to refer to the TCP or UDP stream returning from the target system(s) to the initiator system(s).

**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 in the middle gives 50%/50% chance to Read/Write to be the next IO. Data comparison must be disabled in this mode.

**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 K “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.

**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 saves the contents of the read and write buffers for the transaction. 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. It also saves the contents of the read and write buffers for the transaction and are also extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer. 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. See “-! (or -#) Enable Analyzer trigger writes” on page 240 for additional information.

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**.

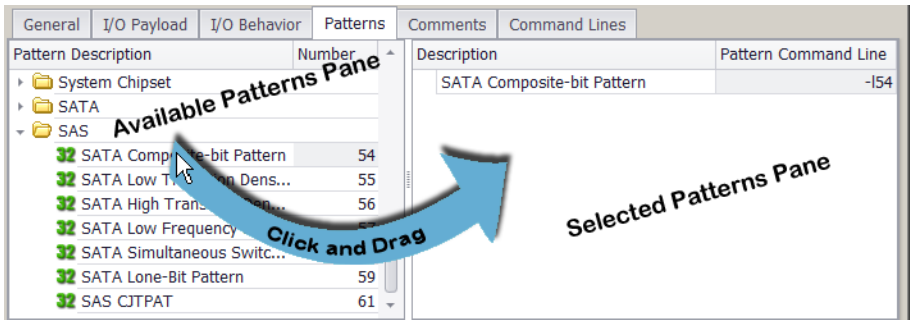
## Patterns Tab

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 65** Available Patterns and Selected Patterns



| Pattern Description | Number |
|---|---|
| System Chipset | |
| SATA | |
| SAS | |
| SATA Composite-bit Pattern | 54 |
| SATA Low T... Dens... | 55 |
| SATA High Tran... | 56 |
| SATA Low Frequency | |
| SATA Simultaneous Switc... | |
| SATA Lone-Bit Pattern | 59 |
| SAS CJTPAT | 61 |

| Description | Pattern Command Line |
|---|---|
| SATA Composite-bit Pattern | -I54 |

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 **Specify Phase Cycle Length** is selected, enter the number of cycle units to run before doing the phase shift. Change the number of units 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. All data patterns that use the **Data Pattern Specification** field default to an all-zero value in the GUI (the length depends on the size of the repeating value data pattern, but it always defaults to all zeros).

The Data Pattern Specification setting corresponds to the CLI data pattern option "-y<hex value>", which does default to the thread number if it is not explicitly set in a CLI command (but only in the CLI). 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 12. The opposing bits are shaded in the table so the walking effect can be seen easily.

**Table 12** 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 the specified number of cycles 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 the blink length by entering a value in the text box or using the numeric spinner.

The following settings applies when the **Compression & Dedup** pattern is selected:

**Random Seed**

**Seed** – Select the check box to specify a 32-bit random number seed value. Enter a random seed value by entering a value in the text box or using the numeric spinner.

**Compression and Deduplication Settings**

**Compression** / **Deduplication** / **Both** radio buttons – Select the option that your testing requires. Select **Compression** for compression testing only; select **Deduplication** for deduplication testing only; or select **Both** for both of the previous options. This selection affects which of the following compression/deduplication options are active.

**Entropy Strength** – Specifies how compressible the payload is.
0 (no entropy, most compressible) to 100 (most entropy, least compressible). It basically defines the percentage of original data that is written after compression is applied. Change the entropy strength by entering a value in the text box or

using the numeric spinner.
For example, when the value is set to 25, this results in data output that is about 25% of the original data size after compression with typical data compression algorithms.
The default value is 100.

**Dedup %** – Specifies the percentage of written data that can be deduplicated by the target device. The percentage of duplicated data blocks can be specified from 0 to 100 percent. Change the dedup percentage by entering a value in the text box or using the numeric spinner. For example, using a value of 30 results in about 70% of generated data being unique and about 30% being filled from a pool of duplicate blocks that can repeated (thus be deduplicated by the target device.) The default value is 0.

**Duplicate Block** – Sets the number of unique blocks in the duplicate block pool to draw from. Change the number of unique blocks by entering a value in the text box or using the numeric spinner. The block correlates with whatever dedup block size that is used by the target's dedup engine and the default block size is set to 8KB. From the second time a block from this pool is written out, it can be deduplicated. The default value is 1.

**Dedup Block Size** – Specifies the dedup block size. It should be set to whatever the value the target storage device's deduplication system uses. Change the block size by entering a value in the text box or using the numeric spinner. The "units" list allows you to select either kilobytes (KB) or megabytes (MB). The default for MLTT is 8 kilobytes. The maximum allowable dedup block size is 1032MB.

### 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 generated 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 GUI Configuration Editors" on page 102.

**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 Tab" on page 160
- "I/O Payload Tab" on page 161
- "I/O Behavior Tab" on page 163
- "Patterns Tab" on page 165
- "Comments Tab" on page 169
- ("Command Lines Tab" on page 169

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 between starting the test and the first sample collection. 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 13** Performance Mode Settings

| GUI Setting | Command Line Setting |
|---|---|
| **I/O Marking and Signing** is set to **No I/O Markings**<br>No I/O signatures are applied to each sector of every write. | `-u` (page 233) |
| **Data Compare Mode** is set to **Disable Data Comparisons**<br>Data comparisons are turned off. | `-n` (page 233) |
| **I/O Behavior** is set to **Keep File Handles Open Between I/Os**<br>Keeps the target file descriptor (handle) open rather than the default behavior of closing and re-opening it after each FOP. | `-o` (page 218) |
| **Use Pattern Reversals** check box is not selected (cleared)<br>Leaves the data patterns reversal after each FOP (forward, then backward) turned off. | `-N` (page 228) |

**Testing Threads** – Set the threads for testing. Each thread executes a single I/O at a time. 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 105 for more information about the **Adding** and **Multiplying** selections.

**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 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 Coordinated Burst Mode, Transactions, I/O Behavior, Triggering, and Error Handling options based on test results.

**Coordinated Burst Mode** – enable this option by selecting the **Enable** check box.

In coordinated burst mode, I/O threads are coordinated such that all threads send their requests to their remote targets at the same time. This can cause the remote peers to simultaneously send back a large amount of data. This can cause the so-called “TCP incast” condition where the switch must drop some incoming packets and end up with severe under-utilization of actually available bandwidth. In coordinated burst mode, the coordinator also waits for all threads to receive their replies from the remote targets before the next burst signal. The completion of one request-response data transfer by all threads combined is a coordinated burst.

Once **Coordinated Burst Mode** is enabled, select either **Stand Alone** or **In a Session**.

In **Stand Alone** mode, the sock process does not coordinate with other sock processes.

While **In a Session** mode, the burst for all coordinators in the session (multiple sock processes from different machines) can be coordinated by a master coordinator. These coordinators must specify a same session ID.

**NOTE**
The master coordinator is designated at the test plan’s run time. The first session listed in the Sockets folder of the test plan is designated as the master.

While **In a Session** mode, slave coordinators signal their I/O threads to begin the burst only when receiving a signal from the designated master coordinator in the same session. A session is designated by a session ID, which is an arbitrary string chosen by the user. A coordinator in a session ignores any communication from coordinators in different sessions. Some notes and restrictions exist for coordinated burst mode:

- Due to the use of broadcast UDP messages, only 1 session controlled sock process can run per host (you can have as many standalone coordinator sock processes per host, though).
- All coordinators (sock) participating in a same session must be in the same subnet (due to the UDP broadcast usage for control messages.) The target systems can be in any subnet as long as they can be reached by sock.
- Slave coordinators will ignore any **Specify Burst Interval** setting since they only wait for the "go" signal from the master.

**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 time between bursts. This selection is available when Coordinated Burst Mode is selected.

**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 saves the contents of the read and write buffers for the transaction. 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 0xCACA-CACA 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 0xDEAD-DEAD 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. It also saves the contents of the read and write buffers for the transaction and are also extremely useful with regard to debugging and analysis. The data value to trigger on occurs in the first two words in the data transfer. 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.See “-! (or -#) Enable Analyzer trigger writes” on page 240 for additional information.

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**.

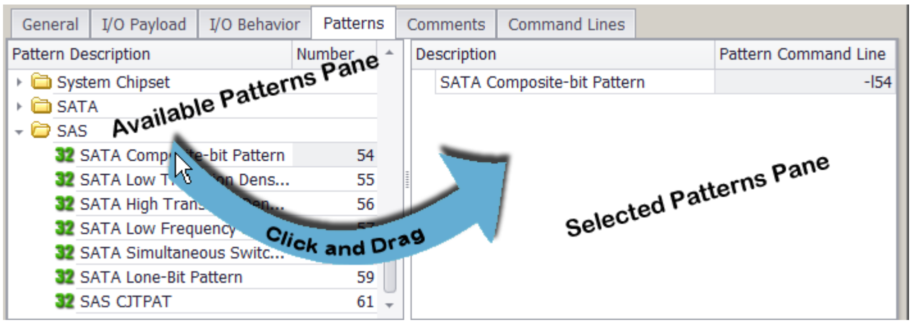
## Patterns Tab

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 66** Available Patterns and Selected Patterns



| Pattern Description | Number |
|---|---|
| System Chipset | |
| SATA | |
| SAS | |
| SATA Comp e-bit Pattern | 54 |
| SATA Low T n Dens... | 55 |
| SATA High Tran n... | 56 |
| SATA Low Frequency | 57 |
| SATA Simultaneous Switc... | |
| SATA Lone-Bit Pattern | 59 |
| SAS CJTPAT | 61 |

| Description | Pattern Command Line |
|---|---|
| SATA Composite-bit Pattern | -I54 |

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 **Specify Phase Cycle Length** is selected, enter the number of cycle units to run before doing the phase shift. Change the number of units 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. All data patterns that use the **Data Pattern Specification** field default to an all-zero value in the GUI (the length depends on the size of the repeating value data pattern, but it always defaults to all zeros).

The Data Pattern Specification setting corresponds to the CLI data pattern option "-y<hex value>", which does default to the thread number if it is not explicitly set in a CLI command (but only in the CLI). 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 14. The opposing bits are shaded in the table so the walking effect can be seen easily.

**Table 14** 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 the specified number of cycles 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 the blink length by entering a value in the text box or using the numeric spinner.

The following settings applies when the **Compression & Dedup** pattern is selected:

**Random Seed**

**Seed** – Select the check box to specify a 32-bit random number seed value. Enter a random seed value by entering a value in the text box or using the numeric spinner.

**Compression and Deduplication Settings**

**Compression** / **Deduplication** / **Both** radio buttons – Select the option that your testing requires. Select **Compression** for compression testing only; select **Deduplication** for deduplication testing only; or select **Both** for both of the previous options. This selection affects which of the following compression/deduplication options are active.

**Entropy Strength** – Specifies how compressible the payload is.
0 (no entropy, most compressible) to 100 (most entropy, least compressible). It basically defines the percentage of original data that is written after compression is applied. Change the entropy strength by entering a value in the text box or

using the numeric spinner.
For example, when the value is set to 25, this results in data output that is about 25% of the original data size after compression with typical data compression algorithms.
The default value is 100.

**Dedup %** – Specifies the percentage of written data that can be deduplicated by the target device. The percentage of duplicated data blocks can be specified from 0 to 100 percent. Change the dedup percentage by entering a value in the text box or using the numeric spinner. For example, using a value of 30 results in about 70% of generated data being unique and about 30% being filled from a pool of duplicate blocks that can repeated (thus be deduplicated by the target device.) The default value is 0.

**Duplicate Block** – Sets the number of unique blocks in the duplicate block pool to draw from. Change the number of unique blocks by entering a value in the text box or using the numeric spinner. The block correlates with whatever dedup block size that is used by the target's dedup engine and the default block size is set to 8KB. From the second time a block from this pool is written out, it can be deduplicated. The default value is 1.

**Dedup Block Size** – Specifies the dedup block size. It should be set to whatever the value the target storage device’s deduplication system uses. Change the block size by entering a value in the text box or using the numeric spinner. The “units” list allows you to select either kilobytes (KB) or megabytes (MB). The default for MLTT is 8 kilobytes. The maximum allowable dedup block size is 1032MB.

### 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 generated 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 GUI Configuration Editors" on page 102.

**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.

The Network CLI Configuration editor only accepts sock commands. For pain and maim commands, refer to "Storage CLI Configuration Editor" on page 146.

### Comments Tab

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

## Format and Secure Erase Configuration Editor

The Format and Secure Erase configuration editor erases the data on a drive leaving it in a clean state after the test process. Using the Format and Secure Erase configuration results in a complete cleaning which means EVERYTHING on the drive will be erased.

To use the "Format and Secure Erase" configuration option select it from the "New Configuration" icon or right-click menu to open the editor. It is also possible to use one of the methods discussed in “Using the GUI Configuration Editors” on page 102.

**NOTE**

The Individual Test Setup settings in the Planning Group Editor (shown in Figure 37 on page 71) and the Test Plan Editor (shown in Figure 38 on page 74) have no effect on the Format and Secure Erase configuration.

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

– “SE Operation Tab” on page 172
– “Command Lines Tab” on page 175
– “Comments Tab” on page 175

### 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:

ATA Devices:

**Security Erase**
**Enhanced Security Erase**

SCSI/NVMe Devices:

**Sanitize Block Erase**
**Sanitize Cryptographic Erase**
**Sanitize Overwrite with Invert=0, Overwrite count =1**
**Sanitize Overwrite with Invert=1, Overwrite count =1**
**Sanitize Overwrite with Invert=1, Overwrite count =2**
**SCSI Format Unit / NVMe Format NVM PI Type=0**
**SCSI Format Unit / NVMe Format NVM PI Type=1**
**SCSI Format Unit / NVMe Format NVM PI Type=2**
**SCSI Format Unit / NVMe Format NVM PI Type=3**

Based on the PI type, you will use the Secure Erase SCSI Device's Format Unit or NVMe Format NVM selections in conjunction with the Testing Style Synchronous (Pain) drop-down list selection on the I/O Payload tab. This table provides guidance with how these settings are used together.

**NOTE**

At the time of the MLTT 7.4.0 release, Windows does not support NVMe Protection Information or metadata

| **Format Unit or Format NVM PI Type** | **Valid Testing Style Selections** | **Invalid Testing Style Selections (applies to SCSI targets, not NVMe targets)** |
|---|---|---|
| PI Type=0 | SCSI /NVMe Passthrough Off<br>SCSI READ/WRITE 10<br>SCSI READ/WRITE 10 + FUA<br>SCSI READ/WRITE 16<br>SCSI READ/WRITE 16 + FUA<br>or<br>NVMe READ/WRITE<br>NVMe READ/WRITE + FUA | SCSI READ/WRITE 32<br>SCSI READ/WRITE 32 + FUA |
| PI Type=1 | SCSI READ/WRITE 10<br>SCSI READ/WRITE 10 + FUA<br>SCSI READ/WRITE 16<br>SCSI READ/WRITE 16 + FUA<br>or<br>NVMe READ/WRITE<br>NVMe READ/WRITE + FUA | SCSI READ/WRITE 32<br>SCSI READ/WRITE 32 + FUA |
| PI Type=2 | SCSI READ/WRITE 32<br>SCSI READ/WRITE 32 + FUA<br>or<br>NVMe READ/WRITE<br>NVMe READ/WRITE + FUA | SCSI READ/WRITE 10<br>SCSI READ/WRITE 10 + FUA<br>SCSI READ/WRITE 16<br>SCSI READ/WRITE 16 + FUA |

| Format Unit or Format NVM PI Type | Valid Testing Style Selections | Invalid Testing Style Selections (applies to SCSI targets, not NVMe targets) |
|---|---|---|
| PI Type=3 | SCSI READ/WRITE 10<br>SCSI READ/WRITE 10 + FUA<br>SCSI READ/WRITE 16<br>SCSI READ/WRITE 16 + FUA<br>or<br>NVMe READ/WRITE<br>NVMe READ/WRITE +FUA | SCSI READ/WRITE 32<br>SCSI READ/WRITE 32 + FUA |

**Note:** If SCSI/NVMe Passthrough OFF is selected for any PI Type, then the OS, not MLTT, handles the metadata and PI.

**Perform full device TRIM or UNMAP if secure erase operation fails** check box enables a fall-back to full-device TRIM command for ATA and UNMAP command for SCSI and NVMe devices if the specified erase command fails. For example, if the **Security Erase** command fails on a SATA drive because it is in Security Frozen state, this option 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.

**Standard erase for ATA, format unit for SCSI, or format NVM for NVMe –**

- 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.
- For NVMe drives, this selection sends a NVMe Format Namespace admin command.

**Enhanced erase for ATA, sanitize for SCSI, or sanitize for NVMe –**

- 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 sends a SCSI sanitize command with the user specifications set.
- For NVMe devices, this selection will request a NVMe sanitize command to be sent with the user specifications set.

**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 generated 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.

## Trim Configuration Editor

The Trim configuration erases specified data blocks. It may be run as a target drive pre-conditioning step before running I/O tests.

Select the "TRIM configuration" option from the "New configuration" icon or right-click menu to open the editor. You may also use one of the methods discussed in “Using the GUI Configuration Editors” on page 102.

**NOTE**

The Individual Test Setup settings in the Planning Group Editor (shown in Figure 37 on page 71) and the Test Plan Editor (shown in Figure 38 on page 74) have no effect on the Trim configuration.

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

- “Trim Operation Tab” on page 176
- “Command Lines Tab” on page 177
- “Comments Tab” on page 177

### Trim Operation Tab

The **Trim Operation** tab allows you to set the parameters for erasing specified blocks from the target drive.

**Threads** – sets the number of threads on the drive 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 drive.

**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 generated 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.

4

# Using the Command Line Switches

This chapter describes the functionality of each command line switch. Topics discussed in this chapter are as follows:

## 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 182)

`\\.\physicaldrive2` is the target device.

`-b512k` is the buffer size (see "I/O Size" on page 184)

`100` is the file size (see "file_size" on page 186).

`-t8` is the thread count (see "Thread Count" on page 188)

`-125` is the data pattern (see "--target-partition Target Partition Range" on page 255)

**IMPORTANT**

You can specify the command line switches in any order. All Medusa Labs Test Tools (MLTT) switches are case sensitive.

## Command Line Switch Conventions

In the **Usage** section of each command line switch, the following conventions are used to convey how each switch is defined:

| | |
|---|---|
| `Courier Font` | denotes commands, options, and separators that must be entered exactly as it is shown. |
| *Italic Font* | denotes a descriptor of required or optional information to be entered for the desired results. |
| < > brackets | denotes that the information within the brackets is required information that must be added to the command. |
| [ ] brackets | denotes that the information within these brackets is optional information that may be added to the command. |

## 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.

### *Data Pattern*

"-I Data Pattern" specifies the desired data pattern number.

### *Online Help*

"-h Online Help" displays the online help.

### *Override Default Session ID*

"-A Override 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 milli-seconds.

### *License Client Operation*

manages 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:***

`-f`*<target>*

***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.

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.

When running pain -m9 or pain -m10 virtual memory target modes, the target can specify one or more NUMA nodes.

For physical targets, the target can specify split write and read device paths which can be useful when testing multi-pathed devices (such as dual-port NVMe controllers with shared namespaces).

For physical and logical targets, appending a partition modifier defines a strict test area boundary within the device. This is useful when assigning different test areas within the same target device to concurrent workloads or test processes.

***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`

NUMA node (for pain -m9 and pain -m10) : `-f1` (where ‘1’ is the NUMA node number)

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 296. 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"
```

Split write/read channels: `-f/dev/nvme0n1:/dev/nvme1n1` (the format is `-f<write_channel>:<read_channel>`)

In the above example, /dev/nvme0n1 and /dev/nvme1n1 are Linux block device paths representing the ports of a dual-port controller with a shared namespace. MLTT writes to the /dev/nvme0n1 path and reads from the /dev/nvme1n1 path and performs data comparison.

Defining a target partition: `-f\\.\physicaldrive1:@10g-200g, -f/dev/sdc:@10g-100g (format is -f<target>:@<area>)`

In the above example, the appended partition specifies an area between byte offsets 10GiB (inclusive) and 200GiB – 1.

If the workload-wide --target-partition is specified, the partition specifier appended to the target specifier overrides the workload-wide specifier.

If both split write/read channels and partition are specified, only one partition specifier must be appended at the end: e.g. `-f/dev/nvme0n1:/dev/nvme1n1:@10g-100g.`

In the following example, first workload performs mapped random-access I/O within the first 60% of the target while concurrently performing non-uniform buffer size sequential-access I/O to the remaining 40% of the same target:

```
pain -%x100 --random-x-map -b4k --full-device -f/dev/nvme0n1:@60%
. -b128k,256k,1m --full-device -f/dev/nvme0n1:@60%-100%
```

Please refer to "--target-partition   Target Partition Range" on page 255 for the partition area syntax.

**CAUTION**

Physical and logical drive access is destructive! Existing data WILL be overwritten.

## I/O Size

### -b   Buffer size

***Usage:***

`-b<buffer_size_set>`

***Description:***

Use `-b` to specify the buffer size for each I/O. This equates to the I/O operation size from the application level. Typically, this value also corresponds to the "transfer size" at the protocol level. The "`<buffer_size_set>`" is a set of one or more size specifiers. If the set contains exactly one size, then that is the uniform buffer size for all I/Os in the workload.

If the set contains more than one size, then, for each I/O, MLTT randomly picks a size from this set of non-uniform buffer sizes.

The “`<buffer_size_set>`” syntax is as follows:

`<size_spec[,<size_spec>[,<size_spec>[,…]]]` where a `<size_spec>` is specified as a discrete value or a range as `[weight@]<buffer_size>[-<buffer_-size>[.alignment]]` where a `<buffer_size>` is `<size[unit]>`

The optional weight has the same meaning as it does for custom read/write mix specification (“`-%`”). Please see “Reads, Writes, And Data Integrity Checking” on page 41 for the explanation on the weight values and how they correspond to percentages. The default weight value, if not specified, is ‘1’.

**NOTE**

For mapped random-access mode, weight values are ignored (i.e. all `<size_specs>` have equal weight.)

In the range form, each I/O picks a random buffer size from the specified range. If an optional alignment value is specified for the range, then all buffer sizes will be multiples of this alignment value. The default alignment value, if not specified, is â€œ1uâ€ (see units below).

`g` = gigabytes

`m` = megabytes

`k` = kilobytes

`b` = bytes (default)

`u` = LB (logical block) size units, usually 512 bytes of the target device (commonly 512 bytes for physical drive, 8 bytes for memory targets, 512 bytes for all other target types).

The default unit, if no unit is specified, is bytes (‘B’).

**NOTE**

For physical drive targets, buffer sizes and alignment values must be multiples of the target device's logical block size (usually 512 bytes.) For memory targets, they must be multiples of 8 bytes. For other types of targets (such as a regular file), they must be a multiple of 512 bytes.

***Examples:***

`-b1m` = 1 megabyte (1048576 bytes.)

`-b4k` = 4 kilobytes (4096 bytes.)

`-b8192` = 8 kilobytes

`-b16k-256k` = For each I/O, use a random buffer size between 16KB and 256KB. Each size is a multiple of default alignment value “1u”.

`-b10@4k,20@16k,70@64k-1m.16k` = 10% of I/Os at 4KB, 20% of I/Os at 16KB, remaining 70% of I/O sizes are randomly picked values between 64KB and 1M where each randomly picked size is a multiple of the specified alignment value, 16KB.

`-b1@1k-16k.1k,2@64k,7@256k` = 1/10 of I/O sizes are randomly picked values between 1K and 16K where each randomly picked size is a multiple of 1KB, 2/10 of I/Os at 64K, and 7/10 of I/Os at 256K

`-b16k,64k,512k` = each I/O randomly picks 16k, 64k, or 512k, where all values have the same probability of having picked.

### *Default:*

The default buffer size in MLTT is 64 kilobytes (65536 bytes).

## 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:***

`-Q`<*queue_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 (1).

## Thread Count

### -t Thread Count

***Usage:***

`-t`<*thread_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 67** Thread Count Command (Threads per Target) Example

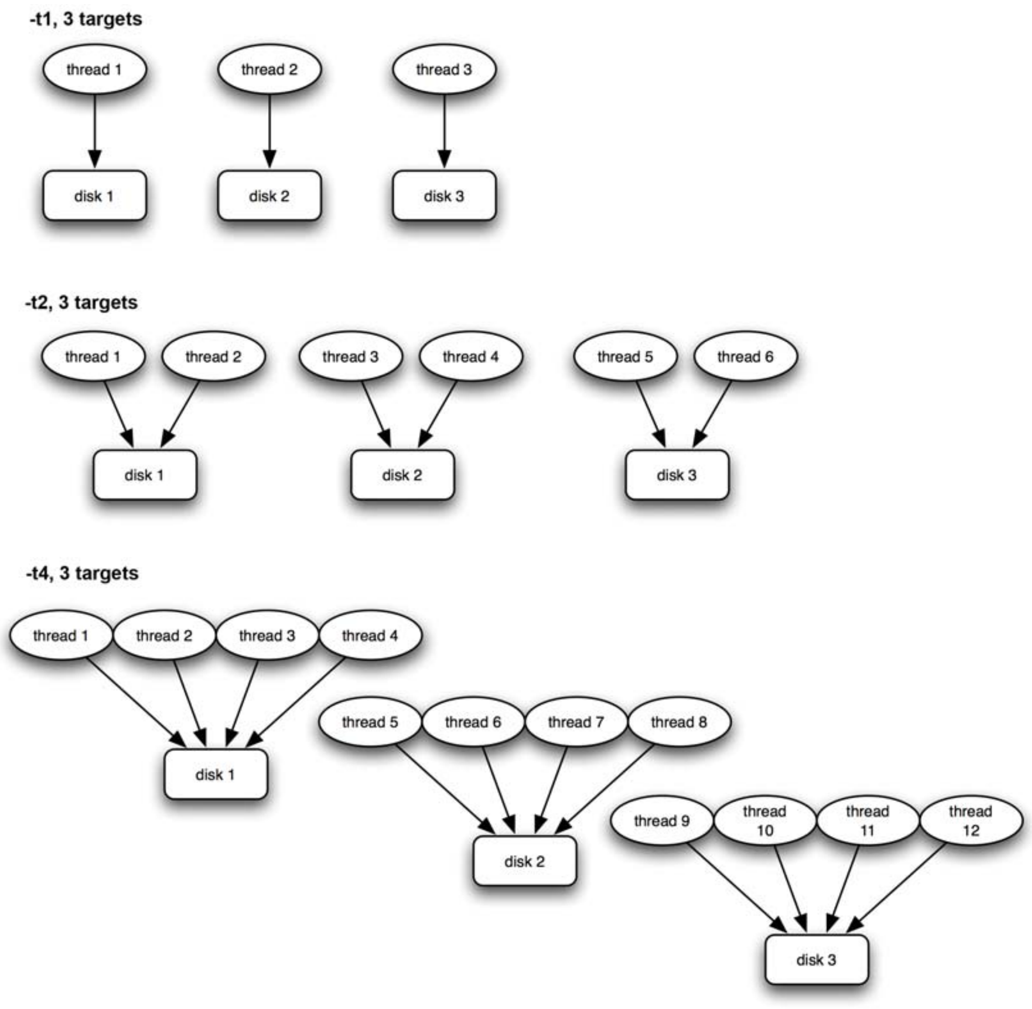
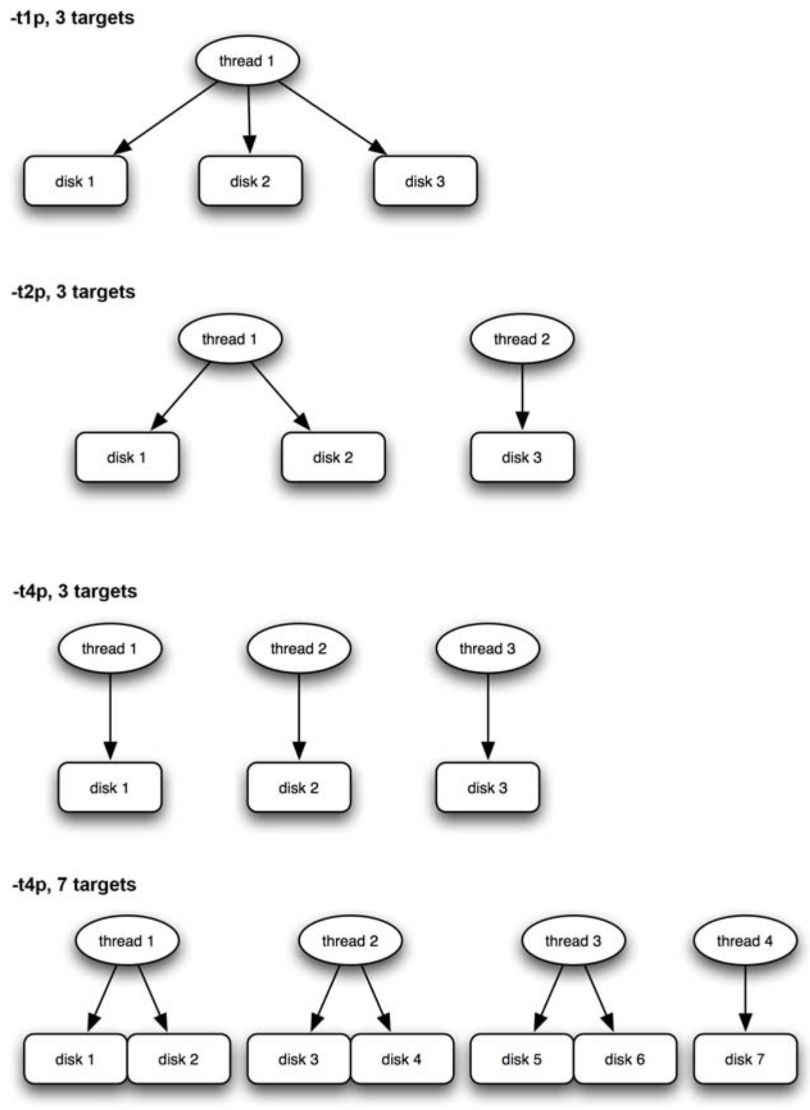


Figure 67 shows examples of normal thread count usage (without 'p' suffix) using 3 targets. For each target, <thread_count> number of I/O threads is created. The total number of I/O threads created in this case is "#targets x <thread_count>".

Figure 68 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 68** 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.

## Concurrent Workloads and Workload Groups

Starting from 7.6.0, MLTT supports running multiple workloads concurrently as well as groups of workloads that run sequentially once the previous group has finished.

A period "." separated list of switch options represents workloads that will run concurrently. A dash "-" separated list of one or more workloads represents workload groups which will run sequentially.

### Concurrent Workloads

It is possible to run multiple workloads at the same time by having a period separated list of workloads. Within each workload, it is possible to assign different switch options such as running each workload to different targets or having each workload run with a different number of threads. There are however, limitations as to what switches are allowed in workloads and it should also be noted that some switches apply to all workloads within a work group.

It is also possible to handle errors using the `--handler` ("--handler Specify Custom Error Handling" on page 283) switch with the `w` option. When errors within workloads are handled this way, if one of the workloads error out, it will not stop the other workloads (i.e. the entire group) or the program but will only stop the workload that had an error.

**NOTE**
Due to the artificial assignment of synchronous and asynchronous I/O to separate executables, MLTT does not allow the mixing of synchronous and asynchronous I/O in a process.

#### Workload Group Global Options

The following list of switches must be placed within the first workload of a workload group and will apply to all workloads within a workload group:

– "-d Test Duration in Seconds" on page 199
– "--latency-histogram Collect Latency Histogram" on page 206
– "-M I/O Monitoring Mode" on page 280

***Examples:***

```
pain -d30 -f/dev/sdc . -f/dev/sdb
```

In the example above, a pain test will run for 30 seconds to both /dev/sdc and /dev/sdb.

```
pain -d30 -f/dev/sdc . -f/dev/sdb - -d60 --latency-histo-
gram=100u,500u,1m,3,5,10 -f/dev/sdc . -f/dev/sdb
```

For this example, there are two workload groups separated by a hyphen “-”. In the first workload group, it will run a pain test for 30 seconds to target /dev/sdc and /dev/sdb. Once that test finishes, it will then move onto the next workload group which is another pain test. For the second workload group however, it will run for 60 seconds and also collect latency histogram data while it is running the pain test to both /dev/sdc and /dev/sdb.

## First Workload of the First Workload Group

### Application Global Options

There are some options the when used will apply to every workload no matter the workload group. These global application options must be specified before the first use of either workload delimiter, “.” or “-” (i.e. the first workload of the first workload group). The following list of command line switches are application global options.

- “--log-utc Coordinated Universal Time(UTC) Timestamps” on page 207
- “-S Seconds to Delay Between Thread Creation” on page 201
- “--sample-delay Specify Sample Delay” on page 203
- “--smart S.M.A.R.T Monitoring” on page 260
- “--x-csv Per-Sample Period CSV Additions” on page 208
- “-Y Seconds Between Performance Samples” on page 203
- --exit-grace-period
- --posix-sigignore
- --target-no-identify
- --tcp-port

### Empty Workload Group

In addition to having the ability to place application global options, the first workload of the first workload group is also the only workload group that is allowed to be empty.

***Examples:***

```
pain . -f/dev/sdc -t3
```

The above command line will run a default pain test for the first group and another pain test with three threads to /dev/sdc. This is valid input

```
pain -f/dev/sdc -t3 .
```

The above command line is not valid since there is an empty workload after the first workload.

```
pain - . -f/dev/sdc -t3
```

The above command line is not valid since the first workload of the second work group is empty.

```
pain - -f/dev/sdc -t3
```

The above command line is valid since the first workload of the first group is empty, which is allowed, and the second workload group does not have any empty workloads.

## Illegal Workload Options

### Non-Workload Options

The following list of switches (and any switch that depends on the specification of these switches) are non-workload options and therefore can only be specified in the absence of workload/workload group delimiters "." and "-".

- "-h   Online Help" on page 204
- "-D   Display the Data Pattern" on page 261
- "--io-trace-parse Parse I/O Traces" on page 212
- "--nvme-get-features Get NVMe Features" on page 223
- "--nvme-set-features NVMe Set Features" on page 224
- "--secure-erase Erase the Target Device and Exit" on page 217
- "--trim Send Trim to Target" on page 219
- "-Z   License Client Operation" on page 217
- --target-check

### Workload Exceptions

The following list of switches (and any switch that depends on the specification of these switches) will result in workload exceptions and can only be specified in the absence of workload/workload group delimiters "." and "-".

- "--journal Run I/O test with journaling enabled" on page 276
- "-V   Reverify Existing Data to a Specified Data Pattern/Verify Journaled Write Operations" on page 272
- -%T (Sock transaction mode) "-%   I/O Profile Specification" on page 235
- -%T: (Sock TCP coordinated burst mode) "-%   I/O Profile Specification" on page 235
- "--cap Limit I/O Throughput" on page 247

## Sequential Workload Groups

In addition to being able to run multiple workloads concurrently, it is also possible to run multiple workload groups sequentially by separating them with a hyphen “-”. When running workloads this way, all workloads of the previous workload group must exit first before the next workload group is started.

It is also possible to handle errors using the `--handler` switch with the `g` option. When errors within the workload group are handled this way, if one of the workload groups error out, it will not stop the entire process (i.e. other workload groups) but will only stop the workload groups that had an error before moving onto the next workload group.

### Rules and Limitations

In addition to all rules and limitations found in “Concurrent Workloads” on page 192, a workload group must contain at least one valid workload specification.

***Examples:***

```
pain -t4 -f/dev/sdb -d10 - -t3 -f/dev/sdc
```

This is a valid command because all workload groups have at least one valid workload.

```
pain - -t3 -f/dev/sdc
```

This is also a valid command because the first workload of the first workload group is the only one that is allowed to be empty, (“First Workload of the First Workload Group” on page 193) making it a valid workload.

```
pain - . -t3 -f/dev/sdc
```

This is not a valid command because the first workload of the second workload group is empty which means that there is an invalid workload in the second workload group.

## 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:

- "Data Pattern Related Switches" on page 261
  - "-D Display the Data Pattern" on page 261
  - "-e Custom Blink Pattern Modifier/Duplicate Block Count" on page 262
  - "-E Custom Blink Pattern Modifier (walking bit variations)/Entropy Strength" on page 263
  - "-F Custom Blink Pattern Modifier" on page 263
  - "-I Invert Pattern Mode" on page 264
  - "-j Data Scrambling Mode" on page 264
  - "-J Data Scrambling Mode Reset Interval" on page 265
  - "-l Specify a Data Pattern Number" on page 265
  - "-L Number of Times to Repeat the Data Pattern Cycle/Continuous Fill/Dedup %" on page 266
  - "-N Disable Data Pattern Reversals" on page 266
  - "-P Modify Data Patterns with a Phase Shift" on page 267
  - "-y Create Data Patterns Based on Various Lengths/Random Seed Value" on page 268
  - "-@ *Read Data Pattern from a File/Deduplication Unit" on page 268*
- "Data Integrity Related Switches" on page 270
  - "-C Comparison Mode" on page 270
  - "-n Disable Data Corruption Checking" on page 271
  - "-u Disable Unique I/O Marks" on page 271
  - "-V Reverify Existing Data to a Specified Data Pattern/Verify Journaled Write Operations" on page 272
  - "--journal Run I/O test with journaling enabled" on page 276
  - "--journal-flush-once Journal Only Flushed Once" on page 277
  - "--jv-compat" on page 278
- "Error Related Switches" on page 279
  - "-H Time to Wait Before Retrying an I/O Operation" on page 279
  - "-M I/O Monitoring Mode" on page 280
  - "-v Verify/Retry Count" on page 280
  - "-! *(or -#) Enable Analyzer trigger writes" on page 281*
  - "--handler Specify Custom Error Handling" on page 283
  - "--reopen-on-retry" on page 285

**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:***

`-d`<*seconds*>

***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:***

`-i`<*iterations*>

***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` affects only the mapped random-access mode (see "--random-x-map Random Access Map" on page 249). For all other types of random-access modes, -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 log Output Level

***Usage:***

`-q`<*mode_number*>

***Description:***

Use `-q` (Quiet Mode) to control the amount of information printed to the screen and log files. The available mode numbers are:

| Mode Number | Description |
|---|---|
| 0 | Standard log file generation/output (default) |

| Mode Number | Description |
|---|---|
| 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 mode number, which is the standard log file generation and output.

## -S Seconds to Delay Between Thread Creation

***Usage:***

`-S<thread_delay>` **or**
`-S.<thread_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.

Use `-T<N>n` as a convenience notation to set the CPU affinity of the I/O threads to the CPUs in the specified NUMA node “<N>”.

***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.`

On a NUMA systems:

```
pain -t8 -T2n
```

In this example, only the CPUs that belong to the second NUMA node will be utilized.

***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 displays the online help.

```
-h -v
```

This example displays the online help for the `-v` option.

## -A Override 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[m]
```

***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. |
| `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. |
| `iops` | Track IOPS for steady state. |
| `mbps` | Track MBPS for steady state. |
| `lat` | Track I/O latency for steady state. |
| %<range_deviation> | Allowed deviation of minimum and maximum tracked values from the average. Default is 20%. |

| | |
|---|---|
| %<slope_deviation> | Allowed deviation of minimum and maximum points in a best linear fit line through the tracked values. Default 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
```

Referring to the following code

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

## --log-utc Coordinated Universal Time(UTC) Timestamps

***Usage:***

```
--log-utc
```

***Description:***

Use `--log-utc` to log all timestamps in UTC rather than in the configured timezone of the DUT host.

***Default:***

Timestamps are logged according to the configured timezone of the DUT host. MLTT records the DUT host's configured time offset from UTC and displays it in the MLTT header.

## --x-csv Per-Sample Period CSV Additions

***Usage:***

```
--x-csv
```

***Description:***

Use `--x-csv` to add minimum, maximum and average I/O completion times for every sample period to the CSV file generated after a test completes Running `--x-csv` in conjunction with `--latency-histogram` will add, track, and update per-sample period latency histogram columns to the CSV file.

## --io-trace I/O Operation History Trace and Payload Data Logging

***Usage:***

```
--io-trace=<r|w|rw>[<,><p1|p2>]
```

***Description:***

Use `--io-trace` to enable examining the history of I/O operations performed by MLTT and, optionally, viewing the payload data associated with each write operation. MLTT creates separate files for I/O operation trace and payload data denoted by a .trace and .datafile extension respectively. The I/O history trace is stored in a compact binary format, named according to the following convention:

`<SESSION_ID>_t<THREAD_ID>-<FILE_SEQUENCE_ID>`.trace

The <*SESSION_ID*> is generated from the start time of the test, prior to I/O generation, encoded as “yymmddHHMMSS”.

<*THREAD_ID*> is the number of the I/O thread.

The <*FILE_SEQUENCE_ID*> is the sequence number of the trace file. As the maximum number of trace events are logged in the trace file, a new trace file with an incremented <*FILE_SEQUENCE_ID*> number will be created.

Payload data files have the same naming convention as the trace files except have a “.data” file extension instead of a “.trace” file extension. These .data files correspond with the `SUBMIT, WRITE` events in the .trace file - see “Trace Events” on page 291 for more information. Each recorded write operation will provide a pointer into the .data file(if any) to locate the associated payload data.

One of tracing options listed below can be enabled by appending it to the end of the command:

| | |
|---|---|
| `r` | Only the read operations will be traced |
| `w` | Only the write operations will be traced |
| `rw` | Both the read and write operations will be traced |

Additionally, one of the payload logging options listed below can be enabled by appending a “,”(comma) after the command followed by the logging option. If a payload logging option is not specified, MLTT will not log the payload data by default:

| | |
|---|---|
| `p1` | Log the entire payload data associated with the write operations |
| `p2` | Log only the first 32 bytes of each LBA of the payload data associated with write operations. This economizes storage space at the expense of added CPU time. This may reduce overall latency between each test I/O compared to the p1 logging option. |

## --io-trace-dir Specify a Directory for I/O Trace

***Usage:***

```
--io-trace-dir=<directory>
```

***Description:***

Use `--io-trace-dir` with `-io-trace` to specify a directory where the trace and data payload file(s) can be created. If the specified directory does not exist, MLTT will attempt to create it. Failure to create the directory will result with a program STARTUP ERROR.

***Default:***

Trace files generated by `--io-trace` will be created in the current working directory of the running MLTT process.

## --io-trace-size Specify Maximum Number of I/O Traces

***Usage:***

```
--io-trace-size=<N>
```

***Description:***

Use `--io-trace-size` with `--io-trace` to specify the maximum number of I/O events that can be logged in a trace file. Once the given number of events are logged, MLTT will create a new .trace file and continue logging.

***Default:***

By default, there are 8192 trace events logged in each .trace file.

## --io-trace-on-error Specify Action on Error for I/O Trace

***Usage:***

```
--io-trace-on-error=<1|2|3>
```

***Description:***

Use `--io-trace-on-error` with `--io-trace` to specify an action to take when a failure occurs during trace file update attempts - e.g. when a file system may become full. Use one of the options below:

| | |
|---|---|
| 1 | Delete the oldest trace file created by the current test and retry. If there has been only one trace file created - i.e. there is no "oldest" trace file to delete - no trace file will be deleted, and retry will be considered a failure. Failure of either the delete operation or the retry operation shall cause the test to terminate. |
| 2 | Ignore trace output error and continue the test if an error is encountered. There will be no retry. Trace output attempts will no longer be made upon a trace output error occurring. |
| 3 | Exit test upon trace output error. There will be no retry. |

***Default:***

The default option is `--io-trace-on-error=1`.

## --io-trace-perf Specify Write Types for I/O Trace

***Usage:***

`--io-trace-perf=<1|2>`

***Description:***

Use `--io-trace-perf` with `--io-trace` to choose either blocking/synchronous or non-blocking/asynchronous writes to the trace files. Use one of the below options:

| | |
|---|---|
| 1 | Use non-blocking/asynchronous writes to trace files. In addition, the writes will be buffered by the OS – this should be more performant especially when the test itself is asynchronous I/O ("maim"). |
| 2 | Use blocking/synchronous writes to trace files. In addition, the writes will bypass OS buffering (e.g. "O_DIRECT" in Linux). This should be used if and only if the test calls for abrupt power-down of the host system |

***Default:***

By default, `--io-trace-perf=1` is assumed.

## --io-trace-parse Parse I/O Traces

***Usage:***

`--io-trace-parse=<filename.trace>`

***Description:***

Use `--io-trace-parse` with `--io-trace` to parse the specified trace file into human-readable format. This is to be run as a separate process after an I/O test ran with `--io-trace`.

***Default:***

By default, this process will automatically concatenate multiple trace files, depending on the `FILE_SEQUENCE_ID`. Typically the first trace file's name should be used - e.g. the one with a `FILE_SEQUENCE_ID` of 1 - so all subsequent trace files will be processed in order.

## --io-trace-output-csv Parse I/O Trace into CSV File

***Usage:***

`--io-trace-output-csv`

***Description:***

Use `--io-trace-output-csv` with `--io-trace-parse` to output the trace data in CSV format. The CSV file created will be in addition to the normal output and will be in the current working directory. The CSV file has the following columns for each event's attributes:

| | |
|---|---|
| `Time` | Event timestamp, microsecond resolution |
| `Event` | S for `SUBMIT`<br>C for `COMPLETE`<br>E for `ERROR` |
| `Op` | R for `READ`<br>W for `WRITE` |

| | |
|---|---|
| `LBA` | I/O LBA in decimal |
| `LBA(hex)` | I/O LBA in hex |
| `Offset` | Byte offset of I/O LBA |
| `Size` | I/O size in bytes |
| `Size(alt)` | I/O size with friendly unit(e.g. `16KB`) |
| `Size(LB count)` | I/O size in LB count |
| `Error` | For `ERROR` events, OS error code if any |
| `Data file` | Data dump file for `SUBMIT,WRITE` |
| `Data offset` | Byte offset of data dump in the data file |
| `Data size` | Byte count of dumped data in the data file |
| `Data` | First 32-byte sample of dumped data |

**NOTE**

To examine all written data corresponding to a `SUBMIT, WRITE` event row in the CSV file, open the data file in a hex viewer application, and examine `Data size` bytes starting from `Data offset`.

## --io-trace-play Streaming a Trace to a Target

***Usage for Medusa Labs Test Tools trace:***

```
<pain|maim> --io-trace-play=mltt:<Trace File>_t<number of
threads>-1.trace -i<number of iterations>
```

**NOTE**

When playing an MLTT trace, the total thread count (`number of threads x target count`) must be less than or equal to the maximum thread count of the source trace.

***Usage for Xgig Trace:***

```
<pain|maim> --io-trace-play=xgig:<Trace File>.txt -i<number of
iterations>
```

***Description:***

Use this switch to stream a trace to a target. The I/Os generated for the source trace can be from anything, i.e., does not have to be from Medusa Labs Test Tools. The following I/O information is streamed to the target as commands: read or write, offset/LBA, and transfer size.

***Default:***

The buffer size is automatically determined during a prescan of the file. If buffer size is specified using -b, it will be ignored unless if the `--io-trace-no-prescan` switch is used in combination with --io-trace-play. If number of iterations is not specified, then it will repeat the play back of the trace indefinitely. There is one thread by default if number of threads is not specified.

**NOTE**

Prescan can be opted out of by appending the "--io-trace-no-prescan Opting out of Initial Prescan" switch option to the command. This is recommend in instances of large trace files as the prescan will take a considerable amount of time for large trace files.

***Example***:

```
pain --io-trace=rw -d1
pain --io-trace-play=mltt:TraceFile.trace_t1-1.trace -i1
```
The first command makes a source trace while the second command will replay the trace. The play back is designated to run with one thread and only run once.

## --io-trace-no-prescan Opting out of Initial Prescan

***Usage:***

```
--io-trace-no-prescan
```

**NOTE**

When using this command, the buffer size ("-b  Buffer size" on page 184) must be specified as there is no prescan to determine the buffer size. The buffer size must be at least equal to the maximum transfer size in the source trace.

***Description:***

This switch is appended to the "--io-trace-play Streaming a Trace to a Target" switch when you want to opt out of the initial prescan. Normally, without this switch, there is an initial prescan of the trace file to determine the maximum transfer length. This switch is recommended for instances of larger trace files as the prescan will take a considerable amount of time in these cases.

***Example:***

```
pain -b64KB --io-trace=rw -d1
pain -b64KB --io-trace-play=mltt:TraceFile.trace_t1-1.trace -i1 --
io-trace-no-prescan
```
The first command makes a source trace while the second command will actually run the trace and record it. The play back is designated to run with one thread and only run once. There is no prescan before the playback so buffer size has to be specified.

## . Running Concurrent workloads

***Usage:***

```
<pain | Maim> <workload 1> [. workload 2 [...]]
```

***Description:***

It is possible to run multiple workloads from the command line concurrently by having a period "." separated list of workloads. With the exception of some switches, each workload can be assigned different switch values to customize each workload. For more information about workload switch limitations and procedure please refer to **"Concurrent Workloads and Workload Groups" on page 192**

***Example:***

```
pain -d30 -t2 -i1 -f/dev/sdb . -t4 -%x100 -f/dev/sdc
```

This is a command which will run two workloads separated by the period delimiter. The options pain and -d30 will be applied to all work loads which means that the first work load and the second workload are set to run a pain test for 30 seconds. The first workload will have 2 threads (`-t2`) run for one iteration (`-i1`) to the target /dev/sdb. In this case of the first workload, it will exit after the first iteration is finished or 30 seconds have passed, whichever happens first.

The second workload will run 4 threads (`-t4`) and run for 30 seconds to the target /dev/sdc in random access mode (`-%x100`).

## - Running Sequential Work Groups

***Usage:***

```
<pain | maim> <group1> [- group2 [...]]
```

***Description:***

It it possible to run multiple groups of workloads in the command line sequentially by using a hyphen delimiter. Within each work group, it is possible to run multiple workloads which results in groups of concurrent work loads that run sequentially. Workload switch limitations still apply within each workload with the exception of pain or maim which will be applied for each work group globally. For more information about work group/workload switch limitations and procedures please refer to **"Concurrent Workloads and Workload Groups" on page 192**

***Examples:***

```
pain -d30 -t1 -f/dev/sdb . -t2 -f/dev/sdc - -d10 -t3 -f/dev/sdb .
-t4 -f/dev/sdc
```

In this example two work groups, each with two work loads, are being ran. In the first work group, each work load will be ran for 30 seconds. The first workload of the first work group will have one thread run to the target, /dev/sdb. Workload two of group one will concurrently run two threads to the target /dev/sdc during this 30 second period.

Once all workloads in group one have exited, work group two will now start. All workloads in group two will run for 10 seconds. The first workload of group two will run 3 threads to the target /dev/sdb while the second workload of group two will concurrently run 4 threads to the target /dev/sdc.

## 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 (for example, `-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.

This option sends a Security Erase command to ATA devices and a Sanitize Block Erase command to SCSI devices. For NVMe devices on Windows, a Format Unit or a Sanitize command is sent through SCSI translation layer. For NVMe devices on Linux, a Format NVM or Sanitize command is sent through the NVMe pass-through IOCTL.

For ATA devices, the "`ata_command`" may be specified immediately after '=' and can be one of the following:

| | |
|---|---|
| 1 | Security Erase |
| 2 | Enhanced Security Erase |

For SCSI/NVMe devices, 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 (T10-PI Type=0) |
| 7 | Format Unit (T10-PI Type =1) |
| 8 | Format Unit (T10-PI Type=2) |
| 9 | Format Unit (T10-PI Type=3) |

Note that the Format Unit selections are only available command for SCSI devices.

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"
```

Based on the T10-PI type, you will use the `--secure-erase` switches in conjunction with the `--scsi` switches. This table provides guidance with how the switches are used together.

| PI Type | Format Command | Valid Commands | Invalid Commands |
|---|---|---|---|
| 0 | --secure-erase=,6 | --scsi=0,1,2,3,4 | --scsi=6,7 |
| 1 | --secure-erase=,7 | --scsi=1,2,3,4 | --scsi=6,7 |
| 2 | --secure-erase=,8 | --scsi=6,7 | --scsi=1,2,3,4 |
| 3 | --secure-erase=,9 | --scsi=1,2,3,4 | --scsi=6,7 |

| PI Type | Format Command | Valid Commands | Invalid Commands |
|---|---|---|---|
| **Note:** Complete `--scsi` switch information is found in “--ptio SCSI/NVME Pass-through I/O Mode” on page 244. `--scsi=0` (off) is valid for all PI types but protection information will not be used or included. | | | |

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. When the command is used, TRIM is sent to a target device and then exits after execution.

For SAS drives, on Windows and Linux, MLTT will send SCSI UNMAP commands via the SCSI pass-through OS function.

For NVMe drives on Windows, MLTT will send SCSI UNMAP commands via the SCSI pass-through OS function. The OS will then translate that to a NVMe Dataset Management NVM command with the Deallocate function and send that to the device.

For NVMe drives on Linux, MLTT will send a NVMe Dataset Management NVM command with the Deallocate function via the NVMe pass-through OS function.

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

## --nvme-erase NVMe Format NVM and Sanitize Administration Commands

***Usage:***

`--nvme-erase=<opt>,<opt>,...`

For Linux Only

***Description:***

Use `--nvme-erase` with `--secure-erase` to NVMe devices to specify NVMe Format NVM and Sanitize admin commands. Options can be a comma-separated list of one or more of the following (case insensitive):

| | |
|---|---|
| `all` | Format NVM common namespace (0xFFFFFFFF) |
| `lbaf:<n>` | Set Format NVM `CDW10:LBAF=<n>` |
| `mset` | Set Format NVM `CDW10.MSET=1` |
| `pil` | Set Format NVM `CDW10.PIL=1` |
| `ses:<n>` | Set Format NVM CDW10:SES=<n> |
| `sanact:<n>` | Set Sanitize `CDW10.SANACT=<n>` |
| `ause` | Set Sanitize `CDW10.AUSE=1` |
| `owpass:<n>` | Set Sanitize `CDW10.OWPASS=<n>` |
| `iopbp` | Set Sanitize `CDW10.IOBPB=1` |
| `ndas` | Set Sanitize `CDW10.NDAS` (No Deallocate After Sanitize) |

| | |
|---|---|
| `ovrpat:<n>` | Set Sanitize `CDW11.OVRPAT` |

## --nvme-get-log NVMe Get Log

***Usage:***

`--nvme-get-log=<`*`page`*`>[,<`*`page`*`>]...`

***Description:***

Use `--nvme-get-log` to retrieve various NVMe log pages. The <*page*> argument may be a comma-separated list of one or more of the following (NOT case-sensitive) in either its decimal, hexadecimal, or mnemonic form:

| | |
|---|---|
| `error (1)` | Error Information |
| `smart (2)` | SMART/Health Information |
| `fwslot (3)` | Firmware Slot Information |
| `nschange (4)` | Changed Namespace List |
| `cse (5)` | Commands Supported and Effects |
| `dst (6)` | Device Self-test |
| `thi (7)` | Telemetry Host-Initiated |
| `tci (8)` | Telemetry Controller-Initiated |
| `endg (9)` | Endurance Group Information (1.4.0) |
| `platn (10)` | Predictable Latency Per NVM Set (1.4.0) |
| `plate (11)` | Predictable Latency Event Aggregate (1.4.0) |
| `ana (12)` | Asymmetric Namespace Access (1.4.0) |
| `pelog (13)` | Persistent Event Log (1.4.0) |
| `lbstat (14)` | LBA Status Information (1.4.0) |
| `endge (15)` | Endurance Group Event Aggregate (1.4.0) |
| `all` | Shortcut to retrieve and output all log pages |
| `all4v` | Like 'all' but only for the device NVMe version |

**NOTE**

MLTT can only distinguish pre-1.4.0 and 1.4.0 or later. If a device implements an NVMe version older than 1.4.0, `all4v` will only skip the log pages introduced in 1.4.0. If the target device, for example, implements 1.0 and if a log page was added in 1.3.0, `all4v` will still attempt to retrieve that log page.

Furthermore, each log page number or the mnemonic name can contain "DOT-separated" options to further specify the CDW10/11/12/13/15 options and general options:

| | |
|---|---|
| `dumpraw` | Save raw binary output to a file (TBD: output file naming scheme) |
| `nsall` | Set Common `NSID (0xFFFFFFFF)` |
| `lsp:<n>` | Set `CDW10.LSP=<n>` |
| `rae` | Set `CDW10.RAE` |

| | |
|---|---|
| `lsid:<n>` | Set `CDW11.”Log Specific Identifier”` (1.4.0) |
| `lpo:<n>` | Set `CDW12.LPOL/VCDW13.LPOU=<n>` |
| `uuid:<n>` | Set `CDW14.”UUID Index”-<n>` (1.4.0) |

See Chapter 5 for examples.

## --nvme-get-features Get NVMe Features

***Usage:***

`--nvme-get-features=<feature>[,<feature>]...`

***Description:***

Use `--nvme-get-features` to retrieve various NVMe feature attributes. The `<feature>` argument may be a comma separated list of one or more of the following (NOT case-sensitive):

| | |
|---|---|
| `arb` | Arbitration |
| `pm` | Power Management |
| `lrt` | LBA Range Type |
| `tt` | Temperature Threshold |
| `er` | Error Recovery |
| `vwc` | Volatile Write Cache |
| `nq` | Number of Queues |
| `ic` | Interrupt Coalescing |
| `ivc` | Interrupt Vector Configuration |
| `wa` | Write Atomicity |
| `aec` | Asynchronous Event Configuration |
| `apst` | Autonomous Power State Transition |
| `hmb` | Hot Memory Buffer |
| `ts` | Timestamp |
| `ket` | Keep Alive Timer |
| `hctm` | Host Controller Thermal Management |
| `nopsc` | Non-operational Power Stat Config |

## --nvme-set-features NVMe Set Features

***Usage:***

```
--nvme-set-features[-sv]=<feature:key=value>[,...]
```

***Description:***

Use `--nvme-set-features` to utilize the NVMe Set Features command. The "`feature`" is one of the values defined for the `--nvme-get-features` option. The `key=value` pairs are feature-specific attributes that can be changed. This option may be specified multiple times, and all specifications will be kept unless the later instance overrides the `feature:key=value` specification of an earlier instance. If the `-sv` suffix is present, the following `feature:key=value` will be saved by the controller (i.e. the "SV" part of the command shall be turned on).

## --nvme-reset-zone ZNS Zone Reset

***Usage:***

```
--nvme-reset-zone=all -f<target>
```

```
--nvme-reset-zone=<ZSLBA> -f<target>
```

***Description:***

This option sends ZNS zone Management Send command with "Reset Zone"action to the specified target. Specify "all" to reset all zones, or specify a Zone starting LBA (ZSLBA) to reset just the specified zone.

## I/O Characteristic Switches

The switches described in this section are the I/O characteristic commands.

### -b Buffer size

***Usage:***

```
-b<buffer_size_set>
```

***Description:***

Use `-b` to specify the buffer size for each I/O. This equates to the I/O operation size from the application level. Typically, this value also corresponds to the "transfer size" at the protocol level. The "`<buffer_size_set>`" is a set of one or more size specifiers. If the set contains exactly one size, then that is the uniform buffer size for all I/Os in the workload. If the set contains more than one size, then, for each I/O, MLTT randomly picks a size from this set of non-uniform buffer sizes.

The “`<buffer_size_set>`” syntax is as follows:

`<size_spec[,<size_spec>[,<size_spec>[,…]]]` where a `<size_spec>` is specified as a discrete value or a range as `[weight@]<buffer_size>[-<buffer_-size>[.alignment]]` where a `<buffer_size>` is `<size[unit]>`

The optional weight has the same meaning as it does for custom read/write mix specification (“`-%`”). Please see “Reads, Writes, And Data Integrity Checking” on page 41 for the explanation on the weight values and how they correspond to percentages. The default weight value, if not specified, is ‘1’.

**NOTE**

For mapped random-access mode, weight values are ignored (i.e. all `<size_specs>` have equal weight.)

In the range form, each I/O picks a random buffer size from the specified range. If an optional alignment value is specified for the range, then all buffer sizes will be multiples of this alignment value. The default alignment value, if not specified, is â€œ1uâ€ (see units below).

`g` = gigabytes

`m` = megabytes

`k` = kilobytes

`b` = bytes (default)

`u` = LB (logical block) size units, usually 512 bytes of the target device (commonly 512 bytes for physical drive, 8 bytes for memory targets, 512 bytes for all other target types).

The default unit, if no unit is specified, is bytes (‘B’).

**NOTE**

For physical drive targets, buffer sizes and alignment values must be multiples of the target device's logical block size (usually 512 bytes.) For memory targets, they must be multiples of 8 bytes. For other types of targets (such as a regular file), they must be a multiple of 512 bytes.

***Examples:***

`-b1m` = 1 megabyte (1048576 bytes.)

`-b4k` = 4 kilobytes (4096 bytes.)

`-b8192` = 8 kilobytes

`-b16k-256k` = For each I/O, use a random buffer size between 16KB and 256KB. Each size is a multiple of default alignment value “1u”.

`-b10@4k,20@16k,70@64k-1m.16k` = 10% of I/Os at 4KB, 20% of I/Os at 16KB, remaining 70% of I/O sizes are randomly picked values between 64KB and 1M where each randomly picked size is a multiple of the specified alignment value, 16KB.

`-b1@1k-16k.1k,2@64k,7@256k` = 1/10 of I/O sizes are randomly picked values between 1K and 16K where each randomly picked size is a multiple of 1KB, 2/10 of I/Os at 64K, and 7/10 of I/Os at 256K

`-b16k,64k,512k` = each I/O randomly picks 16k, 64k, or 512k, where all values have the same probability of having picked.

***Default:***

The default buffer size in MLTT is 64 kilobytes (65536 bytes).

## -B Sequential I/O Direction Control

***Usage:***

`-B`<*mode_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:***

`-g`<*seconds*>

***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:***

`-m`<*mode_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

31 UDP network I/O (use `-f`<ip> or `-f`<hostname> to specify the target peer)

***Default:***

The default mode is 1 for pain, 11 for maim, and 30 for sock.

## -Q Queue Depth (Maim only)

***Usage:***

`-Q`<*queue_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 for Pain and Sock.

***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, and -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 |

***Default:***

The default is `-R2` for Linux; `-R1` for other operating systems.

## -s Single Sector I/O Mode

***Usage:***

`-s`<*sectors*>

***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. This option is ignored for Sock.

## -t Thread Count

***Usage:***

`-t`<*thread_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`. 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 67 on page 189.

Figure 68 on page 190 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 |

***Default:***

The default is `-W2` for Linux; `-W1` for other operating systems.

## -% I/O Profile Specification

***Usage:***

`-%[X:]<spec[,spec[,spec[,...]]]>`

where:

| | |
|---|---|
| X | for weight multiplier |

and where a `spec` is one of following specifiers:

| | |
|---|---|
| `r`[*weight*][@*buffer_size*] | for general mode read percentage or transaction mode request size percentage |

| | |
|---|---|
| `r`[@*buffer_size*] | for coordinated burst mode response size |
| `w`[*weight*][@*buffer_size*] | for general mode write percentage or transaction mode response size percentage |
| `w`[@*buffer_size*] | for coordinated burst mode request size |
| `f`<*weight*> | for forward sequential access percentage |
| b<*weight*> | or backward sequential access percentage |
| `x`<*weight*> | for random access percentage |
| `s`<*random_seed*> | 32-bit seed for I/O profile random number generation |
| o<*size*> | for random offset alignment size to ensure that random I/Os are issued on boundaries of the indicated size |
| `T`[*N*] | for enabling TCP application transaction mode |
| T:[*session_options*] | for enabling TCP coordinated burst mode |
| `z`:<*range1*>[*, range2*[*, ...*]] | for specifying custom access frequency to different areas of the target device. |

***Description:***

The `-%` option can be used to specify a mix of read and write operations, a mix of transfer sizes for read and write operations, and, when running to disk or file targets, I/O access positioning. This is useful in generating I/O modeling real world applications. Furthermore, TCP application transaction mode can be enabled to model general request-response-based application traffic such as HTTP messages between a Web browser and a Web server, while enabling TCP coordinated burst mode can induce “TCP incast” conditions often observed in deployments such as a HADOOP implementation. TCP application transaction mode and coordinated burst mode cannot be used together.

Starting from MLTT 7.6.0, it is possible to have zoned I/O distribution and specify custom access frequencies to different areas of the target device.

Profile specifications may be specified individually or combined with comma. E.g. the following 3 options specifications are equivalent:

```
-%r10@4k,w10@8k,f50,x50,o4k
-%r10@4k,w10@8k -%f50,x50,o4k
-%r10@4k -%w10@8k -%f50 -%x50 -%o4k
```

**General Mode:**

Use `-%r` and `-%w` to specify what percentage of I/O operations should be reads or writes (If neither `-%r` or `-%w` is specified, then both read and write operations will be used). For example, `-%r10 -%w90` (or the equivalent `-%r10,w90`) states “10% read operations, 90% write operations.” The [weight] value is optional for each `-%r` and `-%w` specification and default to value of ‘1’ if not specified. It is recommended to be explicitly set to a desired value. The [@*buffer_size*] modifier is optional and can be used to specify the buffer size used for a `-%r` or `-%w` specification. It can be a single value (e.g. `@64K`) or a MIN-MAX range of random values (e.g. `@4b-64k`). 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. It is also possible to specify multiple read and write multipliers with one `-%` switch e.g.:

```
-%[[read_multiplier],[write_multiplier]]:[r|w]<N>[@<size>],...
```

**NOTE**

If “`-b`” specifies a non-uniform buffer size set, then custom `-%r/w` mix specification is ignored.

Use `-%f`, `-%b`, and `-%x` to specify the probability of I/O direction. Note that in sock, I/O offset and direction are meaningless, but these specifiers are accepted without any effect.

| | |
|---|---|
| `-%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.

For random access, some target devices yield the best performance if the I/O offset is aligned to a certain size. Use `-%o`<*size*> to enforce random access offset alignment. For example, `-%o4k` aligns random access I/O offset to multiples of 4KB.

### TCP Application Transaction Mode (sock-only):

Enable TCP application transaction mode with `-%T`[*N*], where optional value 'N' denotes 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.

In this mode, sock process acts as an application server (e.g. a Web server), while the remote sock targets act as clients (e.g. Web browsers). All I/O per TCP connection between sock and targets is performed in read-write pairs, each pair representing a TCP application-level transaction (e.g. an HTTP GET request from remote paired with a response from sock). Use `-%r` and `-%w` specifiers to specify the percentage and transfer size of request-response data. The syntax for `-%r` and `-%w` are the same as in the general mode, however, in the TCP application transaction mode, you must at least one `-%r` and at least one `-%w` specified.

Per transaction pair, `-%r` represents the client request load while `-%w` represents the server response load. Unlike general mode, the read percentage is determined using only the total weight of `-%r` specifiers, and the write percentage is determined from only the total weight of `-%w` specifiers. At runtime, a read specifier and a write specifier are randomly chose and paired up for each transaction.

`-%f`, `-%b`, `-%x`, and `-%o` are ignored. `-%s` is used for random seed as in the general mode.

**TCP Coordinated Burst Mode (sock-only):**

Enable TCP coordinated burst mode using `-%T:`[*session_options*]. Typically, I/O threads run independently of each other. In coordinated burst mode, I/O threads are coordinated such that all threads send their requests to their remote targets at the same time. This can cause the remote peers to simultaneously send back a large amount of data. This can cause the so-called "TCP incast" condition where the switch must drop some incoming packets and end up with severe under-utilization of actually available bandwidth. In coordinated burst mode, the coordinator also waits for all threads to receive their replies from the remote targets before the next burst signal. The completion of one request-response data transfer by all threads combined is a coordinated burst.

Use `-%r` and `-%w` to specify the request-response data size. The syntax is same as for general mode; however, [*weight*] value is ignored. Also, unlike the general mode, the `-%r` and `-%w` specifiers are not accumulative; only the most recently given `-%r` and `-%w` specifiers are used. In this mode, sock acts as the client while the remote targets act as the servers. Per request-response, `-%w` represents the request from sock to remote targets, and `-%r` represents the response from the remote targets.

If neither `-%r` nor `-%w` is specified, write-read coordinated burst transactions are performed using the buffer size specified by `-b`.

`-%f`, `-%b`, `-%x`, `-%o,` and `-%s` are ignored.

Use `-g`<*delay*> to pause the coordinator in between bursts

A sock process can act as a stand-alone coordinator using -%T: without session options, e.g.:

```
        sock -%T:,w100b,r128k -t16
```

In stand-alone mode, the sock process does not coordinate with other sock processes.

On the other hand, multiple sock processes from different machines can be in a session where a master coordinator coordinates the burst for all coordinators in the session. These coordinators must specify a same session ID, and one must be designated as the master with the “m” modifier:

Master coordinator sock process with session ID “session1”:

`sock -%T:m:session1` …

Slave coordinator sock processes joining “session1”:

`sock -%T:session1` …

In the session mode, slave coordinators signal their I/O threads to begin the burst only when receiving a signal from the designated master coordinator in the same session. A session is designated by a session ID, which is an arbitrary string chosen by the user. A coordinator in a session ignores any communication from coordinators in different sessions. Some notes and restrictions exist for coordinated burst mode:

- Due to the use of broadcast UDP messages, only 1 session controlled sock process can run per host (you can have as many stand-alone coordinator sock processes per host, though).
- All coordinators (sock) participating in a same session must be in the same subnet (again, due to the UDP broadcast usage for control messages.) The target systems, of course, can be in any subnet as long as they can be reached by sock
- If you first start the master with "`-%T:m:xxx`", the master runs its own bursts by itself until other coordinators start with "`-%T:xxx`". You can gradually ramp up the number of slave coordinators, for example, and they'll get picked up by the master coordinator.
- If you start a slave coordinator first, it waits until the master of the session starts up.
- Slave coordinators ignore `-g` settings since they only wait for the "go" signal from the master.
- If you quit the master coordinator sock process, the slaves will quit as well.

***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 specifying equal distribution for both reads and writes by not specifying the “r” or “w” operation token. It also demonstrates the split read/ write multiplier.

| | |
|---|---|
| `-%1,2:1@1k,3@2k` | “1@1k,3@2k” is the compact equivalent of “r1@1k,w1@1k,r3@2k,w3@2k”. Split multiplier “1,2” scales the read weights by 1 and write weights by 2. Therefore, after applying the multipliers, the entire specification is the equivalent of “-%r1@1k,r3@2k -%w2@1k,w6@2k”. |

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.” |

The following example demonstrates the transaction mode:

```
sock -%T20-30 -%r100@30b-500b,w20@4k,w20@8k,w60@100k
```

In this 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.

The following examples demonstrate the coordinated burst:

```
sock -%T:,w100b,r128k -t16
```

In this coordinated burst mode example, a stand-alone coordinator coordinates its 16 I/O threads for coordinated bursts of 100 byte request and 128KB response data.

Master: `sock -%T:m:x -b32k -t16 -ftargets.dat`

Slave: `sock -%T:x -b32k -t16 -ftargets.dat`

These two examples are of multi-coordinator session with session ID “x”.

All threads send 32KB or data and receive 32KB of data. Each thread waits for the next coordination signal when the send-receive transaction is done. When all threads finish the send-receive transaction, the coordinator sends the signal for the next burst.

```
sock -%T:x -b32k -t16 -ftargets.dat -w
```

This uses the write-only option “-w” to make it a send-only burst.

```
sock -%T:x -b32k -t16 -ftargets.dat -%w
```

This is also send-only: -%w without the @buffer_size modifier.

```
sock -%T:x -b32k -t16 -ftargets.dat -%w@100b -g5
```

Send only, but using buffer size of 100 bytes (overrides -b) - also coordinator pause of 5 seconds after the end of each burst ("end of burst" is when all I/O threads are done with the burst).

```
sock -%T:x -t16 -ftargets.dat -%w@100b,r@32k
```

This example uses unequal send-receive buffer sizes of sending 100 bytes then receiving 32K in request-response pairs. It is probably the most representative case for TCP incast where a coordinator sends small requests to multiple systems which then respond with larger data and cause the switch to drop packets.

## Zoned I/O Distribution

Starting from MLTT 7.6.0, it is possible to customize specific access frequency to different areas of the target device. Each zone specification may contain one or more range specifications and apply to all targets specified for a workload. When attempting to assign different zone specifications to different targets, different workloads or different workload groups are required.

**NOTE**

It is possible to use multiple `-%z:` switches per workload.

***Usage:***

```
-%z:<range specification 1>[,range specification 2[,...]]
```

### *Range Specification:*

```
[r|w]<N>@<area specification>
```

When specifying ranges, you can specify whether it will be read only, write only or both and then assign it a weight (denoted by the value N).

Every range specification is associated with an operation:

| | |
|---|---|
| r | Signifies that the specified area will be read only. |
| w | Signifies that the specified area will be write only. |
| None Specified | Signifies that the specified area will be read and write. |

The denoted weight determines what percentage of an operation occurs in an area.

For example the range specification:

```
-%z:r1@10%, r3@50%
```

Means that 25% of the reads occur in the first 10% of the device while the other 75% of the reads occur in the next 50% of the device. This is because the total of the weights added together is 4. In the first area specified, it is 1/4 which means that 25% of the reads will occur in the area specified (First 10% of the device). In the second area specified, it is 3/4 which means that 75% of the reads will occur in the area specified (50% of the device after the first 10%)

### *Area Specification:*

Area specification on the device can be specified with one of the following notations:

"Next X" Notation: `<X>[unit]`

In this notation, X is an integer value and "unit" is the same size unit recognized by MLTT for other size or range operations. The very first "Next X" area notation specified is the "First X" area of the target.

"Span" Notation: `<begin>[unit] -[end][unit]`

In this notation, a specific area of the device is specified for operation from "`begin`" to "`end`". The beginning of the range is included for writing while the ending of the range is excluded for writing.

**NOTE**
Only the final range specification per "`-%z`" may specify open-ended "end" range.

"Remainder" Notation: Blank

If this notation is used, it means that the final unassigned area of the specified target area will be used. This notation is the equivalent of having an open-ended “span” notation.

**NOTE**
Only the final range specification per “`-%z`” may specify this notation.

In any of the above notations, if a unit is specified, the default unit used is “B”

In any of the above notations, the actual calculated target offset values shall be truncate-aligned to the target logical block size.

***Rules and Restrictions:***

The `-%z` option shall be accepted if-and-only-if “100% random-access” is specified. (i.e. specifying “`-%z<...>`” with a random-access specification that combines “`-%f`” and/or “`-%b`” shall be an error)

The “`-%z`” option implies a read/write I/O profile mode (meaning, 100% read, 100% write, or some mix of read and write). If no “`-%r/w`” option is specified and no 100% read (-r) or 100% write (-w) option is given, “-%r50,w50” shall be assumed.

The “`-%z`” option implies shared starting offset (“`-X0`”). Any “`-X`” explicitly specified by the user shall be ignored.

The “`-%z`” option implies full-device access.

Overlapping zones is not allowed and will result in an error.

***Examples:***

```
-%z:r1@10%,r1@50%
```

50% of the reads shall occur in the first 10% of the specified target area. The other 50% of the reads shall occur in the next 50% of the specified target area. No operations will occur in the final 40% of the specified target area.

```
-%z:r1@10%,r1@50%,r2@40%

-%z:r1@10%,r1@50%,r2@-

-%z:r1@10%,r1@50%,r2@

-%zr1@10%,r1@50%,r2
```

All of the above examples are functionally equivalent. In these examples, 25% of the reads occur in the first 10% of the specified target area. The next 25% of the reads occur in the next 50% of the specified target area. The last 50% of the reads occur in the final 40% of the specified target area.

```
-%z:r1@10%,r1@1m-1g,r1@-5g,r1@100g- \

-%z:w1@10%,w1@1m-1g,w1@-5g,w1@100g
```

In the above example, 25% of the reads and writes occur in the first 10% of the specified target area. The next 25% of the reads and writes occur in between byte offsets 1m and 1g (not including byte offset 1g). 25% of the reads and writes shall occur between byte offsets 1g and 5g (not including byte offset 5g). The last 25 of the reads and writes shall occur from byte offset 100g to the end of the specified target area. No reads or writes occur in the area between 5g and 100g

```
-%z:1@10%,1@1m-1g,1@-5g,1@100g
```

The above example is functionally equivalent to the example above it. Omitting the operation ‘r’ or ‘w’ from the specification token defines equal zones for both read and write operations.

## --ptio SCSI/NVME Pass-through I/O Mode

***Usage (for SCSI):***

`--ptio[=0|1|2|3|4|5|6|7]`

***Description:***

Use direct SCSI commands for read/write. This option is ignored if not running synchronous I/O to physical drive targets. The Read/Write selections are available with and without Forced Unit Access (FUA) as noted below. The available mode numbers are:

| | |
|---|---|
| **SCSI Passthrough Off** | Not using direct SCSI command. |
| **READ/WRITE 10** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 10 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **READ/WRITE 16 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI types 0,1, and 3. |
| **SCSI UNMAP / ATA TRIM** | Supports the TRIM command. A more complete description is provided below. |
| **READ/WRITE 32** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |

| **READ/WRITE 32 + FUA (Forced Unit Access)** | Incorporates Protection Information in the SCSI command when device formated to T10-PI type 2. |
|---|---|

| | |
|---|---|
| 0 = Off (default) | Not using direct SCSI command. |
| 1 = On (READ10/WRITE10) | Supports `--secure-erase` SCSI commands using Format Unit T10-PI types 0, 1, and 3. |
| 2 = On (READ10/WRITE10), with FUA | Supports `--secure-erase` SCSI commands using Format Unit T10-PI types 0, 1, and 3. |
| 3 = On (READ16/WRITE16) | Supports `--secure-erase` SCSI commands using Format Unit T10-PI types 0, 1, and 3. |
| 4 = On (READ16/WRITE16), with FUA | Supports `--secure-erase` SCSI commands using Format Unit T10-PI types 0, 1, and 3. |
| 5 = SCSI UNMAP / ATA TRIM | Supports the TRIM command. A more complete description is provided below. |
| 6 = On (READ32/WRITE32) | Supports `--secure-erase` SCSI commands using Format Unit T10-PI type 2. |
| 7 = On (READ32/WRITE32), with FUA | Supports `--secure-erase` SCSI commands using Format Unit T10-PI type 2. |

If '`--ptio`' is specified without using '0', '1', '2', '3', '4', '5', '6', or '7', then '`--ptio=1`' is assumed.

If the target device has one of the protection types enabled in the "--secure-erase Erase the Target Device and Exit" command, this command interacts with those protection type selections.

– If the target device has Type 1 or Type 3 PI enabled, you must select one of the following commands: `--ptio=1`, `--ptio=2`, `--ptio=3`, or `--ptio=4`

– If the target device has Type 2 PI enabled, you must select one of these commands: `--ptio=6` or `--ptio=7`

The `--ptio=5` command turns on SCSI UNMAP (or ATA TRIM) as a write operation for normal I/O engine tests. It overrides any `-l` setting with `-l69` (all zeros) and it turns on the "-u Disable Unique I/O Marks" command. Reads are done using the normal operating system functions, while writes are replaced with UNMAP or TRIM to the requested offset using buffer size. The assumption is that most devices will return buffers filled with "0's" for reads after TRIM (without any write in-between).

***Default:***

```
--ptio=0
```

***Usage (For NVME):***

```
--ptio=<1|2|3|4|5|6|7>
```

For Linux Only

***Description:***

Send NVMe pass-through commands.

Listed below are the options for <n>:

| | |
|---|---|
| `1,3,6` | Sends NVMe Read and Write NVM commands via NVMe pass-through IOCTL. For ZNS targets, ‘6’ sends Zone Append Command. |
| `2,4,7` | Sends NVMe Read and Write NVM commands with FUA via NVMe pass-through IOCTL. For ZNS targets, ‘7’ sends Zone Append Command. |
| 5 | Sends a NVMe Dataset Management function Dellocate NVM command via NVMe pass-through IOCTL |

Generic metadata transfer will be supported if the LBAF in use has metadata. Extended LBA and metadata in a separate buffer will be supported. PI information will be supported if the LBAF in use has PI enabled.

For Windows NVMe targets, this mode will continue to send the specified SCSI READ/WRITE commands via SCSI-NVMe translation.

## --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.

## --nvme-io NVMe Commands

***Usage:***

```
–nvme-io=<opt>,<opt>,...
```

For Linux Only

***Description:***

Use `--nvme-io` with `–ptio=<1,2,3,4,6,7>` to further specify command options as a comma-separated list of 1 or more of the following:

| | |
|---|---|
| `lr` | Set `CDW12.LR=1` (limited retry) |
| `pract` | Set `CDW12.PRINFO.PRACT=1` (for PI) |
| `prchk.guard` | Set CDW12.PRINFO.PRCHK enable PI Guard Field check |
| `prchk.app` | Set CDW12.PRINFO.PRCHK enable PI app tag check |
| `prchk.ref` | Set CDW12.PRINFO.PRCHK enable PI ref tag check |
| `prchk.all` | Shortcut to enable checking all PI fields |

## --random-x-map Random Access Map

***Usage:***

`--random-x-map <-m17|-%x<N>> [-i<N>]`

***Description:***

By default, MLTT random-access I/O does not keep track of accessed LBAs. This results in previously accessed LBAs being accessed again in full or in part while some LBAs might never be accessed during a test.The `--random-x-map` option enables unique LBA random-access tracking which enables using `-i` iteration limit and other options, such as `-V`, which require knowing if the test has accessed all LBAs within the specified test area.

**NOTE**

"`--random-x-map`" does not enable random-access I/O. It enables unique LBA tracking for random-access I/O. Specifying "`--random-x-map`" for sequential-access I/O has no impact (i.e. this option is ignored for sequential-access I/O). For "`--random-x-map`" to have an effect, the test must also enable random-access mode with options such as the "`-m17`" macro or an appropriate "`-%x`" option. Please refer to "MLTT Basics" on page 22.

***Rules and Limitations:***

- "100% random-access" must be specified. Attempting to use `--random-x-map` with a random-access specification ("`-m17`" or "`-%x<N>`") that combines "`-%f`" and/or "`-%b`" will result in an error.
- Specifying both `--random-x-map` and `-%o<N>` where `<N>` is different from the uniform buffer size for the workload will result in an error.
- Specifying both `--random-x-map` and zoned I/O distribution (`-%z:<...>`) will result in an error.

## --io-repeat I/O Repeat Count

***Usage:***

`--io-repeat=<N>`

***Description:***

This switch allows specifying the number of times an I/O is performed at a given offset (LBA) before performing I/O at the next offset.

## 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:***

`-f`<*target*>

***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.

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.

When running pain -m9 or pain -m10 virtual memory target modes, the target can specify one or more NUMA nodes.

For physical targets, the target can specify split write and read device paths which can be useful when testing multi-pathed devices (such as dual-port NVMe controllers with shared namespaces).

For physical and logical targets, appending a partition modifier defines a strict test area boundary within the device. This is useful when assigning different test areas within the same target device to concurrent workloads or test processes.

***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`

NUMA node (for pain -m9 and pain -m10) : `-f1` (where ‘1’ is the NUMA node number)

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 296. 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"
```

Split write/read channels: `-f/dev/nvme0n1:/dev/nvme1n1` (the format is `-f<write_channel>:<read_channel>`)

In the above example, /dev/nvme0n1 and /dev/nvme1n1 are Linux block device paths representing the ports of a dual-port controller with a shared namespace. MLTT writes to the /dev/nvme0n1 path and reads from the /dev/nvme1n1 path and performs data comparison.

Defining a target partition: `-f\\.\physicaldrive1:@10g-200g, -f/dev/sdc:@10g-100g (format is -f<target>:@<area>)`

In the above example, the appended partition specifies an area between byte offsets 10GiB (inclusive) and 200GiB – 1.

If the workload-wide --target-partition is specified, the partition specifier appended to the target specifier overrides the workload-wide specifier.

If both split write/read channels and partition are specified, only one partition specifier must be appended at the end: e.g. `-f/dev/nvme0n1:/dev/nvme1n1:@10g-100g.`

In the following example, first workload performs mapped random-access I/O within the first 60% of the target while concurrently performing non-uniform buffer size sequential-access I/O to the remaining 40% of the same target:

```
pain -%x100 --random-x-map -b4k --full-device -f/dev/nvme0n1:@60%
. -b128k,256k,1m --full-device -f/dev/nvme0n1:@60%-100%
```

Please refer to “--target-partition   Target Partition Range” on page 255 for the partition area syntax.

**CAUTION**

Physical and logical drive access is destructive! Existing data WILL be overwritten.

## --file-per-thread Create Target Files for each Thread

***Usage:***

```
[-t<N>] [file size] -f<target file> --file-per-thread
```

***Description:***

By default, when a regular file is specified as the target, and if the specified thread count N is greater than 1, all I/O threads run to the same specified "`target file`" with each thread assigned its own "`file sized`" I/O area within the "`target file`" whose size is "`N x file size`". However, if the `--file-per-thread` option is also specified, all threads sharing the same "`target file`" each instead run to their own "`target file.x`" where x is the thread number and each target file is "`file sized`". This option is applicable only to file system targets. If the target is a physical device or a logical volume, `--file-per-thread` is ignored.

**NOTE**
`--file-per-thread` is the default behavior when "`-f<target file>`" is not specified. Without the explicitly specified target, `pain/maim` uses the default "`-f<HOSTNAME> --file-per-thread`" options.

***Default:***

This switch option is automatically enabled when no target is specified (even without being called).

This switch option is disabled by default if an explicit target is given (but can be included to override default behavior).

***Examples***:

```
pain -t2
```

Each thread creates its own target file of given file size (in this case, no file size is given, so 4MB default `pain` is used).

- Thread 1 -> "<HOSTNAME>.0001" 4MB file
- Thread 2 -> "<HOSTNAME>.0002" 4MB file

```
pain -t2 --file-per-thread
```

Same as above (`--file-per-thread` is redundant in this case).

```
pain -t2 -fmytargetfile
```

Both threads run to the same 8MB "`mytargetfile`", each in its own 4MB area within it.

- Thread 1 -> first 4MBs of “`mytargetfile`”
- Thread 2 -> next 4MBs of “`mytargetfile`”

```
pain -t2 -fmytargetfile --file-per-thread
```

Each thread uses its own target file of 4MBs.

- Thread 1 -> “`mytargetfile.0001`” 4MB file
- Thread 2 -> “`mytargetfile.0002`” 4MB file

```
catapult -f pain -t2
```

`Catapult` executes: `pain -t2 -ftargetfile.dat`

- Thread 1 -> first 4MBs of “`targetfile.dat`”
- Thread 2 -> next 4MBs of “`targetfile.dat`”

```
catapult -f pain -t2 --file-per-thread
```

`Catapult` executes: `pain -t2 -ftargetfile.dat --file-per-thread`

- Thread 1 -> “`targetfile.dat.0001`” 4MB file

Thread 2 -> “`targetfile.dat.0002`” 4MB file

## --target-partition Target Partition Range

***Usage:***

```
--target-partition=<area>
```

***Description:***

This option specifies the partition modifier to apply to every target in the Workload. Specifying a partition on a target defines the absolute boundaries of the test area within the target device, and MLTT enforces that all I/Os are confined within the specified partitioned area of the target.

The “`<area>`” specification uses the simplified zoned I/O area syntax (see “`-%z:`”).

- “First X” notation: first “`<size>[unit]`” bytes of the target device.
- “Span” notation: “`<begin>[unit]-<end>[unit]`”

The optional "`[unit]`" is the same as it is for the file size parameter: 'k' for kilobytes, 'm' for megabytes, '%' for percentage of device size, etc. The default unit, if not specified, is 'm'.

In the "span" notation, the open-ended "`<end>`" implies "to the rest of the device".

**NOTE**

The partition defines the test area boundaries, not the test area itself.

While this option specifies the partition of all targets of a Workload, the "`-f`" option can specify per-target partitions. If both "`--target-partition`" and per-target partitions are specified, per-target partition overrides the "`--target-partition`". See "-f Target" on page 182 for examples of specifying per-target partitions.

Examples:

`--target-partition=50%` Define a partition of first 50% of the device size on all target devices in the Workload

`--target-partition=2g-16g` Define a partition in the device range between byte offsets 2GB (inclusive) and 16GB

`--target-partition=75%-` Open ended span example – partition is from the 75% of the device size to the end of the device

`--target-partition=50% -f\\.\physicaldrive1:@75%` Override example. The per-target partition specifier of 75% overrides the Workload `–target-partition=50%`

## -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[0|1|2|3|4|5]`

### *Description:*

Use `-O` to override the MLTT default device base offset setting. The mode definitions are:

| | |
|---|---|
| `0` | Override base offset, error out on bad I/O extent |
| `1` | Override base offset, adjust bad I/O extent |
| `2` | Override base offset, allow bad I/O extent |
| `3` | Do not override base offset, error out on bad I/O extent |
| `4` | Do not override base offset, adjust bad I/O extent |
| `5` | Do not override base offset, allow bad I/O extent |

If `-O` is specified without an optional mode value, then `-O0` is assumed.

This option also sets the policy for checking I/O attempt beyond the device extent.

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.

This option is ignored for 'sock' or 'pain/maim' file system targets.

***Default:***

`-O3` where the I/O starts at 1MB default offset, and the I/O settings beyond device size error out early.

## -x Starting Offset

***Usage:***

`-x`*<offset>*

***Description:***

Use `-x` to specify the starting offset. This mode allows multiple host systems or multiple sessions of the tools on a single system to access the same device or file concurrently by allowing the user to specifying different starting offsets such that the multiple I/O streams do not overlap.

You can specify the starting offset number in any units including “u”. If no unit suffix is provided, the default value of “m” (megabytes) is assumed. 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.)

This option is ignored for sock.

***Default:***

The default offset, if `-x` is specified without the optional offset value, is 0MB.

## -X Shared Offset Mode - All Threads Issue I/Os to the Same Offsets

***Usage:***

`-X`*<offset>*

***Description:***

Use the `-X` switch to specify shared offset mode. In this mode, all threads issue I/Os to the same offsets. You can specify the starting offset number in any units including “u”. If no unit suffix is provided, the default value of “m” (megabytes) is assumed.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.

This option is ignored for sock.

***Default:***

The default offset, if `-X` is specified without the optional offset value, is 0MB.

## --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`”.

This option is ignored for sock.

***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*]

***Description:***

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. This is supported for both ATA and NVMe target devices. See Chapter 5 - SMART Log for more information and examples.

***Default:***

Unset

## Data Pattern Related Switches

This switch category contains switches that you use when you specify patterns for your test.

- "-D Display the Data Pattern" on page 261
- "-e Custom Blink Pattern Modifier/Duplicate Block Count" on page 262
- "-E Custom Blink Pattern Modifier (walking bit variations)/Entropy Strength" on page 263
- "-F Custom Blink Pattern Modifier" on page 263
- "-I Invert Pattern Mode" on page 264
- "-j Data Scrambling Mode" on page 264
- "-J Data Scrambling Mode Reset Interval" on page 265
- "-l Specify a Data Pattern Number" on page 265
- "-L Number of Times to Repeat the Data Pattern Cycle/Continuous Fill/Dedup %" on page 266
- "-N Disable Data Pattern Reversals" on page 266
- "-P Modify Data Patterns with a Phase Shift" on page 267
- "-y Create Data Patterns Based on Various Lengths/Random Seed Value" on page 268
- "-@ *Read Data Pattern from a File/Deduplication Unit" on page 268*

For more information about data patterns, refer to Chapter 6 "Data Pattern Reference".

### -D Display the Data Pattern

***Usage:***

```
-D[*[b]|[bytes]
```

***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 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. 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/Duplicate Block Count

***Usage:***

`-e`<*bit_length*|*duplicate_block_count*>

***Description:***

The `-e` switch is a modification option for the custom blink pattern variations for `-l99` or the duplicate block count for `-l80`.

For `-l99`, this switch sets the 'length' of blinking 'on' bits. This switch can be used in conjunction with the `-L` switch to create some interesting patterns across various bus lengths. Refer to "-e Length of Blinking Bits" on page 262 for a detailed usage example.

For `-l80`, this switch is used with the `-L` switch to define deduplication data characteristics. This option is ignored for `-L0`. Refer to "Deduplication/Compression Pattern (-l80)" on page 264 for complete information related specifically to `-l80`.

***Default:***

Set to '`-L`' value for '`-l99`'; set to '-e1' for '`-l80`'.

## -E Custom Blink Pattern Modifier (walking bit variations)/ Entropy Strength

***Usage:***

`-E`<*hold_cycles*|*entropy_strength*>

***Description:***

The `-E` switch is a modification option for the custom blink pattern with walking bit options (`-l99w, -l99o, -l99f`) or the compression entropy setting for `-l80`.

For `-l99`, 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 Set Custom Blink Hold Cycles (before transition for bit-walking variations)" on page 263 for a detailed usage example.

For `-l80`, this switch specifies how compressible the payload is. 0 (no entropy, most compressible) to 100 (most entropy, least compressible). It basically defines the percentage of original data that is written after compression is applied. For example, `-E25` results in data output that is about 25% of the original data size after compression with typical data compression algorithms. Refer to "Deduplication/Compression Pattern (-l80)" on page 264 for complete information related specifically to `-l80`.

***Default:***

'100' for `-l80`; unset for other patterns.

## -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 Reset custom blinking pattern to the initial pattern value each cycle" on page 263 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 258 for a detailed usage example of this switch.

***Default:***

This option is disabled by default.

## -j Data Scrambling Mode

***Usage:***

`-j`<*number*>

***Description:***

Use `-j` to enable prescrambling 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:***

`-J`<*bytes*>

***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 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 data pattern is `-l17` (16-bit incrementing/decrementing pattern) for pain, maim or sock.

## -L Number of Times to Repeat the Data Pattern Cycle/ Continuous Fill/Dedup %

#### *Usage:*

`-L`*<count|fill_type|dedup%>*

#### *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.

For pattern `-l0` (Read pattern from data file), specify `-L2` for continuous fill or `-L3` for continuous fill with reset at each FOP.

#### *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 257 for a detailed usage example.

For pattern `-l80`, this switch specifies the percentage of written data that can be deduplicated by the target device. The percentage of duplicated data blocks can be specified from 0 to 100 percent. For example, using `-l80 -L30`, about 70% of generated data will be unique and about 30% will be filled from a pool of duplicate blocks that can be repeated (thus be deduplicated by the target device.) Refer to "Deduplication/Compression Pattern (-l80)" on page 264 for complete information related specifically to `-l80`.

For all other patterns, repeat each unit of a data pattern 'count' times.

#### *Default:*

The default value is `-L0` for `-l0` and `-l80`; `-L1` for other patterns.

## -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 255 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:*

`-P`<*unit_shift_interval*>

#### *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 unit interval, such that the square wave, created by the blinking on/off bits reverses. You can specify frequency of the phase shift as the number of units to run before reversing or a default value is used if you enter `-P` only.

#### *Example:*

`pain -l14-P4` would cause a shift every 4 units of this 64-bit based data pattern. Refer to “-P Modify Data Patterns with a Phase Shift” on page 258 for an in-depth discussion on the use of this switch and a detailed usage example.

#### *Default:*

The default length varies by pattern.

## -y Create Data Patterns Based on Various Lengths/Random Seed Value

***Usage:***

`-y`<*pattern_value*|*random_seed_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 (such as `-l1`, `-l2`,and `-l4`) to create patterns based on various lengths.

For data patterns involving random number generation (such as `-l35`, `-l47`, `-l60`, `-l62`, and `-l80`), this option may be used to specify a 32-bit random number seed value.

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 269 for more information on this switch.

***Default:***

A default value is assigned that is equivalent to the thread index number.

## -@ Read Data Pattern from a File/Deduplication Unit

***Usage:***

`-@`<*path/file_name*|*dedup_unit*>

***Description:***

For `-l0`, this switch identifies the path and file name that contains the data pattern that will be used by the `-l0` data pattern (a special data pattern that uses a user-specified file to fill the pattern). The file must be greater than or equal in size to 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. Refer to "Specified Data Patterns" on page 269 for complete information and an example related specifically to `-l0`.

For `-l80`, this switch specifies the dedup block size. It should be set to whatever the value the target storage device's deduplication system uses. The default for MLTT is 8K.

**NOTE**

You can use a size suffix like 'k', '`b`', etc. as usual, but it will not accept the "u" suffix. "k" is the default suffix (i.e. `-@32` is `-@32K`). This option is ignored for `-L0`. Refer to "Deduplication/Compression Pattern (-l80)" on page 264 for complete information related specifically to `-l80`.

## 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:***

`-C`*<compare_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/Verify Journaled Write Operations

***Usage:***

`-V` Reverify existing data to a specified data pattern

`--journal -V` Verify journaled write operations

### Reverification of existing data to a specified data pattern

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 (or, preferably, the "`-Vw`" option). 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.

**NOTE**

For random-access I/O, "-V" supports only the mapped random-access mode (see "--random-x-map Random Access Map" on page 249)

***Examples:***

Write a data pattern (note that -i1 is specified):
```
pain -l99 -L16 -i1 -w 100 -fg:\test.dat
```

Or, with the preferred -Vw option:
```
pain -l99 -L16 -Vw 100 -fg:\test.dat
```

Read the pattern back from a backup location and verify data for 1 iteration
```
pain -l99 -L16 100 -V -fh:\test.dat
```

Default:

This option is disabled by default.

**NOTE**

The data re-verification option (-V) should not be used with data pattern 46 (`-l46`) because the buffer memory address is likely to have been re-assigned between test runs.

## Verification of Journaled Write Operations

The verification process looks for the journal log in the current working directory. You can use `--journal<=path/directory>` to specify the location of the journal log. To avoid issues with locating the journal log, the best practice is to always specify a path/directory for journaling and verification. When you run the journal verification (`--journal -V`), the following summary information is output to the screen and the log file at the end of the test:

```
JOURNAL VERIFICATION SUMMARY
    TOTAL (both queued and confirmed writes):
        240
    CONFIRMED (confirmed writes):
        227
    VERIFIED (queued or confirmed writes that passed):
        239
    FAILED (confirmed writes with data corruption):
        0
    SKIPPED (skipped verification due to overwrite by newer write(s)):
        0
    UNDEFINED (queued writes that failed verification):
        1
```

***Example:***

In addition, the log file also lists the details for each recorded write operation that was verified. Here are examples of three records:

```
JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:0
    JOURNAL: OFFSET    : 4377804800 (0x0000000104F00000)
    JOURNAL: LBA       : 8550400 (0x0000000000827800)
    JOURNAL: SIZE      : 64KB
    JOURNAL: LOOP      : 0
    JOURNAL: TIMESTAMP : 0x51AC51A7E2B9E010
    JOURNAL: WRITE     : QUEUED
    JOURNAL: VERIFY    : OK


JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:0
    JOURNAL: OFFSET    : 4377673728 (0x0000000104EE0000)
    JOURNAL: LBA       : 8550144 (0x0000000000827700)
    JOURNAL: SIZE      : 64KB
    JOURNAL: LOOP      : 0
    JOURNAL: TIMESTAMP : 0x51AC51A7E2B9E00E
    JOURNAL: WRITE     : CONFIRMED
    JOURNAL: VERIFY    : OK


JOURNAL: Target:\\.\physicaldrive1 Thread:1 CTX:15
    JOURNAL: OFFSET    : 11024896 (0x0000000000A83A00)
    JOURNAL: LBA       : 21533 (0x000000000000541D)
    JOURNAL: SIZE      : 64KB
    JOURNAL: LOOP      : 0
    JOURNAL: TIMESTAMP : 0x51AC55D16454B001
    JOURNAL: WRITE     : CONFIRMED
    JOURNAL: VERIFY    : SKIP (OVERWRITE @ 0x51AC55D16BF5D000)
```

In these examples:

"OFFSET:", "LBA:", and "SIZE:" detail the I/O location and transfer size.

"LOOP:" is the sequential I/O loop count during which the recorded write occurred.

"TIMESTAMP:" is a concatenation of 52-bit microsecond time stamp and 12-bit sequence number to uniquely identify a recorded write. This number is unique within each I/O context ("CTX:").

"WRITE:" status can be "QUEUED" or "CONFIRMED". Before each write, the record for the write is set to "QUEUED" state and committed to the journal. When the write operation completes, the record for the write is set to "CONFIRMED" state and committed to the journal.

"VERIFY:" status is set during journal verification as each recorded data is read back and a data integrity check is performed on it.

The status indicators are set based on the following definitions:

| | |
|---|---|
| **`OK`** | if "CONFIRMED" or "QUEUED" write passes data integrity check. |
| **`FAIL`** | if a "CONFIRMED" write fails data integrity check. This also raises the regular "Data corruption" error. |
| **`SKIP`** | if the pain/maim detects that the I/O location was overwritten by a later write operation, and data verification is skipped in order to avoid falsely raising a data corruption error. In addition, the "TIMESTAMP" corresponding to over-writing I/O is given next to the status. |
| **`UNDEFINED`** | if a "QUEUED" write (i.e. queued but not confirmed write operation) fails data integrity check. Because the write was not confirmed, this is not flagged as a data corruption error. |

***Default:***

This option is disabled by default.

## -Vw Write-once for Reverification

***Usage:***

`-Vw`

***Description:***

This switch is a convenience macro for "`-w -i1`". It is intended to be used to write data that can then be verified with the `-V` switch in a later command that has the exact same switches, except with the `-Vw` switch replaced with the `-V` switch for verification.

***Example:***

```
maim -l80 -t3 -Q7 --full-device -%x100 --random-x-map -
b1k,2k,4k,128k-1m.4k -f\\.\PhysicalDrive2 -Vw
```

Then for verification, run:

```
maim -l80 -t3 -Q7 --full-device -%x100 --random-x-map -
b1k,2k,4k,128k-1m.4k -f\\.\PhysicalDrive2 -V -i3
```

## --journal Run I/O test with journaling enabled

***Usage:***

`--journal`<=*path/directory*>

***Description:***

Use the `--journal` switch to record a log file of recent write operation characteristics (buffer size, thread count, queue depth, file size, pattern used, etc.). Then, when a power loss to the initiator or the target device is simulated during testing, the log file is saved. The log file preserves the status of the last several write operations. A path and directory can be specified as a location to save the .log file. Note that you may use relative path or absolute path for the journal directory, however, the specified directory must already exist.

When the power is restored, run the `--journal -V` switches together to retrieve the saved log file and verify the last known completed write operations prior to the power loss. The `--journal -V` switches ignore all other options (with the exception of "`-f`"), but to be safe, specify no other options. However, if a path and directory was identified in the initial save portion of the process, likewise, the same path and directory must be identified during the retrieval portion.

Pain/maim restores the recorded command line options and initializes with the recorded write operation characteristics. Then it reads the journal log file for the recorded write operations, reads the data back from the target devices at the recorded LBA by recorded transfer size, and does the normal data comparison to see if the write data was correctly committed to the device.

***Example:***

maim -Q64 -t4 -b8k --journal=/my/journal/directory

For verification mode, specify:
maim -V --journal=/my/journal/directory

If you specify an optional journal directory during journaled I/O, you must specify the same directory during journal verification mode.

You can specify a raw disk as the journal directory. For example:

| | |
|---|---|
| On Windows: | --journal=\\.\PhysicalDrive1 |
| On Linux: | --journal=/dev/sdb |

Journal data is written starting at 1MB offset of the specified disks. Using a raw disk can reduce the journal data update latency and better ensure the journal data integrity.

## --journal-flush-once Journal Only Flushed Once

***Usage:***

```
--journal-flush-once
```

***Description:***

Use `--journal-flush-once` with `--journal` to have MLTT not perform the normal two journal commits per write operation. Instead, the entire in-memory journal data will be written once at the end of the test. This is often used in instances where instead of removing power to the entire DUT host, only the DUT is powered-off. By removing the 2-per-write synchronous journal file commits, the user can perform journaled tests at the near-maximum performance capability of the DUT.

## --jv-compat

***Usage:***

```
--jv-compat
```

***Description:***

Apply the pre-7.4.0 ambiguity resolution method for journal verification. Please see Appendix H for further explanation.

## 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:***

`-H`<*seconds*>

***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 280.

***Default:***

By default, retries are performed immediately.

## -M I/O Monitoring Mode

***Usage:***

`-M`*<seconds>*

***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:***

`-v`*<retry_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 279.

This option is ignored for sock.

***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:***

`-![0|1|2|3|4|5]` or `-#[0|1|2|3|4|5]`

***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. 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:

| | |
|---|---|
| `-!0` | No trigger |
| `-!1` | Writes `0xCACACACA 0xCACACACA` for data corruption, `0xCACACACA 0xDEADBEEF` for I/O error trigger |
| `-!2` | Writes `0xDEADDEAD 0xDEADDEAD` for data corruption, `0xDEADDEAD 0xDEADBEEF` for I/O error trigger |
| `-!3` | Exits on error immediately - no trigger written |
| `-!4` | Executes external command specified in 'MedusaTools.cfg |
| `-!5` | Writes default `0xCACACACA` trigger and exits immediately |

If '`-!`' is specified without 'mode', '`-!1`' is assumed.

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;

**NOTE**

TriggerAnalyzer.cmd and StopAnalyzer.cmd are batch files that are included with the Medusa Labs Test Tools installation for Windows. These files are typically installed in this folder: C:\Program Files\Medusa Labs\Test Tools\bin

Ensure that the batch files, TriggerAnalyzer.cmd and StopAnalyzer.cmd, and the executable, wget.exe, are in the executable path. A mirror for downloading the executable, wget.exe, can be downloaded from the URL below:

http://wget.addictivecode.org/FrequentlyAskedQuestions.html#download

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 Appendix D in the *Xgig Analyzer User Guide*.

### *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 I/O Monitoring Mode” on page 280) |
| `‘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 informational ('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><w\|g>[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 '-!'. Starting from MLTT 7.6.0, it is possible to handle errors for workloads ('w') or workload groups ('g'). |
| `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 '-! |

**CAUTION**

If data comparisons are being performed on write or read commands, it is recommended that you not use the `--handler` label spec (`l<i|w|e>` listed above) to change the label of the write or read commands to either informational ('`i`') or warning ('`w`'). This would result in these errors being ignored.

Example: `pain --handler=read,write:tr,xip,pBAD,v3`

On read or write errors, regular trigger to target ('tr') using trigger pattern 'BADBADBAD-BADBADB' ('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' will be ignored with these settings because these events would no longer be treated as errors. 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 '-!'

## --reopen-on-retry

***Usage:***

```
--reopen-on-retry
```

***Description:***

Use the `--reopen-on-retry` switch to close and open a new file descriptor before retrying a failed I/O. This option is ignored for 'sock'.

**NOTE**

It is recommended that this switch be used in any test cases that involve path failover in multi-path I/O configurations. It is possible for the file descriptor to be lost when a failover occurs and I/O traffic may not recover unless a reopen on the file or device is performed.

***Default:***

Unset (do not reopen before retry).

5

# Logging and Output

This chapter describes the log files used in Medusa Labs Test Tools Suite. MLTT provides detailed logs of performance and error conditions. Topics discussed in this chapter are as follows:

## Overview

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 70 on page 300 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), the minimum, maximum, and average MB/s, and the minimum, maximum and average I/O response times. A count of total errors encountered is also listed. Catapult uses this file to verify test results.

The following shows an example of a performance summary log.

```
[Test Info]
          Command Line = pain 4MB -b512B -i1 -Y1 -M5
          I/O Size = 512B
          Queue Depth = 1
          Test Mode = Read write mix
          Start Time = 10/14/2015 11:27:12 AM
[Completion Info]
          Elapsed Time = 00:00:07
          Samples = 7
          I/O Halts = 0
          Avg Completion Time in Seconds = 0.000400
          Max Completion Time in Seconds = 0.032027
          Min Completion Time in Seconds = 0.000225
[I/O Operations]
          Total IOs = 17374
          Avg IO/Sec = 2482.00
          Max IO/Sec = 3178.75
          Min IO/Sec = 1924.85
[Megabytes]
          Total MB = 8
          Avg MB/Sec = 1.21
          Max MB/Sec = 1.55
          Min MB/Sec = 0.94
[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 is installed with MLTT and can be found in this path: "C:\Program Files\Medusa Labs\Test Tools\bin". 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 over-written. 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. This batch file can be is installed with MLTT and can be found in this path: "C:\Program Files\Medusa Labs\Test Tools\bin". 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 spreadsheet applications 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. The .csv log file has the following 38 column headings for each performance sample:

| | | | | | |
|---|---|---|---|---|---|
| **1** | Sample Number | **14** | Min Completion Time | **27** | Current % CPU |
| **2** | Elapsed Time | **15** | I/O Halts | **28** | Avg % CPU |
| **3** | FOPS | **16** | I/O Timeouts | **29** | Max % CPU |
| **4** | Current Avg IOPS | **17** | Write Errors | **30** | Min % CPU |
| **5** | Avg IOPS | **18** | Read Errors | **31** | Current % CPU User Only |
| **6** | Max IOPS | **19** | Data Corruptions | **32** | Avg % CPU User Only |
| **7** | Min IOPS | **20** | Open Errors | **33** | Max % CPU User Only |
| **8** | Current Avg MB/s | **21** | Seek Errors | **34** | Min % CPU User Only |
| **9** | Avg MB/s | **22** | Flush Errors | **35** | Current % CPU Interrupt Only |
| **10** | Max MB/s | **23** | Close Errors | **36** | Avg % CPU Interrupt Only |
| **11** | Min MB/s | **24** | Remove Errors | **37** | Max % CPU Interrupt Only |
| **12** | Avg Completion Time | **25** | Size Errors | **38** | Min % CPU Interrupt Only |
| **13** | Max Completion Time | **26** | Retry Errors | | |

Specifying `--x-csv` will add per-sample period minimum, maximum, and average I/O completion times. Specifying `--x-csv` and `--latency-histogram` will update the per-sample period latency histogram columns.

## Error Log

The error log 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. Any error codes returned by the operating system are also included and decoded.

# I/O Operation History Trace and Payload Data Logging

This section goes into more detail about the contents of the I/O history traces and the data dump files - see --io-trace in Chapter 4 for the command line switches.

## Trace Events

Inside each traced event the following attributes are recorded:

**Event:** `SUBMIT, COMPLETE, or ERROR`

**Operation:** `READ or WRITE`

**Time delta:** Elapsed time since the last traced events in microseconds

**LBA:** Logical block address of I/O location

**Status Transfer Size or Error Status (if OS error):**

`SUBMIT` Event - The I/O size in logical block count

`COMPLETE` Event - The returned I/O size in logical block count - i.e. the actual I/O size of a completed I/O claimed by the OS.

`ERROR` Event for OS Error - This is an OS error code.

`ERROR` Event for Data Corruption - The completed transfer size claimed by the OS in logical block count.

`ERROR` Event for Transfer Size Mismatch - The completed transfer size claimed by OS in logical block count.

`ERROR` Event for MLTT Application-level Precondition Check Failure - This will always return 0 .

**NOTE**

Due to the ambiguity in trying to interpret the value of this field, if the requested and returned size are the same for a read operation then the ERROR is likely data corruption. Cross reference the other MLTT logs - i.e. .bad and .log files - to see the error details.

Each traced operation records two events in the form of either a `SUMBIT-COMPLETE` pair or a `SUMBIT-ERROR` pair. Each event captured in the .trace file will be one of the following pairs below depending on the parameters specified - see `--io-trace` in Chapter 4 for more information.

`SUBMIT, WRITE` - This event is traced before a `WRITE` operation is submitted to the OS.

`SUBMIT, READ` - This event is traced before a `READ` operation is submitted to the OS.

Example of `SUBMIT, WRITE` without data dump or `SUBMIT, READ`:

```
SUBMIT, WRITE @ 2019/05/23-14:07:31.292.839
lba : 12288 (0x3000) offset : 6291456
size : 1048576 (1MB, 2048 LBs)
```

Where each field is:

| | |
|---|---|
| `Event, Operation @ timestamp of event` | |
| `lba` | I/O LBA in decimal and hex |
| `offset` | Byte offset of I/O LBA |
| `size` | Requested I/O size in bytes and LB count |

Example of `SUBMIT, WRITE` with data dump:

```
SUBMIT, WRITE @ 2019/05/23-13:13:42.579.532
lba : 24576 (0x6000)
offset : 12582912 size : 1048576 (1MB, 2048 LBs)
dfile : /test/190523131307_t1-25.data
doffset : 3145728 dsize : 1048576
data : FFFFFFFE FFFDFFFC 64640001 ED00000C 00006000
 FFF5FFF4 FFF3FFF2 FFF1FFF0
```

Where each field is:

| | |
|---|---|
| `Event, Operation @ timestamp of event` | |
| `lba` | I/O LBA in decimal and hex |
| `offset` | Byte offset of I/O LBA |
| `size` | Requested I/O size in bytes and LB count |
| `dfile` | The corresponding data dump file where the write data was written |
| `doffset` | The byte offset into the data dump file where the write data was written |
| `dsize` | The byte count of data written to the data dump file |
| `data` | The first 32-bytes sample of the written data |

**NOTE**
To examine the entire written data corresponding to this event, open the “dfile” in a hex viewer application and then examine “dsize” bytes starting from “doffset”.

`COMPLETE, WRITE` - This event is traced after MLTT receives a “success” status from the OS for a previously submitted `WRITE` operation.

`COMPLETE, READ` - This event is traced after MLTT receives a “success” status from the OS for a previously submitted `READ` operation.

Example of a `COMPLETE` event:

```
COMPLETE, WRITE @ 2019/05/23-13:13:42.722.104
lba : 2048 (0x0800)
offset : 1048576 size : 1048576 (1MB, 2048 LBs)
```

Where each field is:

| | |
|---|---|
| `Event, Operation @ timestamp of event` | |
| `lba` | I/O LBA in decimal and hex |
| `offset` | Byte offset of I/O LBA |
| `size` | Requested I/O size in bytes and LB count |

`ERROR, WRITE` - This event is traced after MLTT receives a status from the OS for a previously submitted `WRITE` operation and the returned status is “error”, an MLTT application-level precondition check failed before submitting the `WRITE` operation, or the returned status is “success”, but the requested I/O size and the returned I/O size do not match.

`ERROR, READ` - This event is traced after MLTT receives a status form the OS for a previously submitted `READ` operation and the returned status is “error”, an MLTT application-level precondition check failed before submitting the `READ` operation, the returned status is “success”, but the requested I/O size and returned I/O size do not match, or the returned status is “success”, but data corruption was detected.

Example of an data corruption or transfer size mismatch `ERROR` event, `READ` or `WRITE`:

```
ERROR, READ @ 2019/05/23-14:49:09.609.616
lba : 2048 (0x0800)
offset : 1048576
size : 1048576 (1MB, 2048 LBs)
```

Where each field it:

| | |
|---|---|
| `Event, Operation @ timestamp of event` | |
| `lba` | I/O LBA in decimal and hex |
| `offset` | Byte offset of I/O LBA |
| `size` | Requested I/O size in bytes and LB count |

Example of an OS error “ERROR” event, `READ` or `WRITE`:

```
ERROR, READ @ 2019/05/23-15:20:21.161.245
lba : 20480 (0x5000)
offset : 10485760
error : 5
```

Where each field is:

| | |
|---|---|
| `Event, Operation @ timestamp of event` | |
| `lba` | I/O LBA in decimal and hex |
| `offset` | Byte offset of I/O LBA |
| `size` | Requested I/O size in bytes and LB count |

## NVMe Identify and NVMe Get Log

This sections contains sample output and logs generated by MLTT specifically for NVMe devices.

### NVMe Identify

When running any command to an NVMe drive, MLTT will automatically output the NVMe controller and namespace identifications of the DUT to the console (if being run from the command line) and the .log file output. Below is an example of a DUT’s output.

```
NVMe Admin Identify: Controller (CNS=1, win32Status=0, CQE.DW0=0, CQE.DW3.SF.SCT=0, CQE.DW3.SF.SC=0)
    VID: 32902 (0x8086)
    SSVID: 32902 (0x8086)
    SN: "PHHH852204AS128A"
    MN: "INTEL SSDPEKKA128G8                 "
    FR: "005D"
    RAB: 6 (0x06)
    IEEE: 6083300 (0x5CD2E4): "5C-D2-E4"
    CMIC: 0 (0x00): single NVM port, single controller, PCI Function or Fabric
    MDTS: 6 (0x06)
    CNTLID: 1 (0x0001)
    VER: 66304 (0x00010300): "1.3.0"
    RTD3R: 500000 (0x0007A120)
    RTD3E: 2000000 (0x001E8480)
    OAES: 512 (0x00000200): Firmware Activation Notices
    OACS: 23 (0x0017): Security Send and Receive, Format NVM, Firmware Commit and Image Download, Device
Self-test
    ACL: 4 (0x04)
    AERL: 7 (0x07)
    FRMW: 20 (0x14): first slot read-write, slot count 2, firmware activation without reset
   LPA: 15 (0x0F): SMART/Health Information per namespace, Commands Supported and Effects, extended data
for the Get Log Page, Telemetry Host-Initiated and Controller-Initiated
    ELPE: 255 (0xFF)
    NPSS: 4 (0x04)
    AVSCC: 0 (0x00): vendor-specific format
    APSTA: 1 (0x01): autonomous power state transition
    WCTEMP: 348 (0x015C)
    CCTEMP: 353 (0x0161)
    MTFA: 50 (0x0032)
    EDSTT: 5 (0x0005)
    DSTO: 1 (0x01): one per NVM subsytem at a time
    FWUG: 0 (0x00)
    HCTMA: 1 (0x0001): supported
    MNTMT: 303 (0x012F)
    MXTMT: 348 (0x015C)
    SANICAP: 3 (0x00000003): crypto erase, block erase
    SQES: 102 (0x66): minimum 6, maximum 6
    CQES: 68 (0x44): minimum 4, maximum 4
    NN: 1 (0x00000001)
    ONCS: 95 (0x005F): Compare, Write Uncorrectable, Dataset Management, Write Zeroes, Save and Select
Features, Timestamp
    FNA:4 (0x04): format specified namespace, secure erase specified namespace, cryptographic erase
    VWC: 1 (0x01): present
    AWUN: 0 (0x0000)
    AWUPF: 0 (0x0000)
    NVSCC: 0 (0x00): vendor-specific format
    MNAN: 0 (0x00000000)
    SUBNQN: "nqn.2017-12.org.nvmexpress:uuid:11111111-2222-3333-4444-555555555555"
    PSD0: MP 900, MXPS 0 (0.01W), RRT 0, RRL 0, RWT 0, RWL 0
    PSD1: MP 460, MXPS 0 (0.01W), RRT 1, RRL 1, RWT 1, RWL 1
    PSD2: MP 380, MXPS 0 (0.01W), RRT 2, RRL 2, RWT 2, RWL 2
    PSD3: MP 450, MXPS 1 (0.0001W), NOPS, ANLAT 2000, EXLAT 2000, RRT 3, RRL 3, RWT 3, RWL 3

    NVMe Admin Identify: Namespace (CNS=0, win32Status=0, CQE.DW0=0, CQE.DW3.SF.SCT=0, CQE.DW3.SF.SC=0)
    NSZE: 250069680 (0x000000000EE7C2B0)
    NCAP: 250069680 (0x000000000EE7C2B0)
    NUSE: 250069680 (0x000000000EE7C2B0)
    NLBAF: 0 (0x00)
    FLBAS: 0 (0x00): LBA Format 0
    NMIC: 0 (0x00): attach to single controller
    FPI: 128 (0x80): supported, %remaining 0
    DLFEAT: 1 (0x01):  deallocated logical block all bytes set to 0x00, PI Guard field of deallocated
logical block is 0xFFFF
    NAWUN: 0 (0x0000)
    NAWUPF: 0 (0x0000)
    NACWU: 0 (0x0000)
    NGUID: 0x0000000001000000E4D25CC1456A5001 ("00000000-0100-0000-e4d2-5cc1456a5001")
    LBAF0: MS 0 (0x0000), LBADS 9 (0x09), RP 0 (0x00) Best performance
```

This is the output found from the `nvme-cli` for the commands `nvme id-ns` and `nvme id-ctrl` in Linux.

## NVMe Get Log

This section contains example output of the NVMe Get Log feature. Multiple get logs may be specified by putting a “,”(comma) between them and MLTT will section off each get log by its log id. To retrieve a get log, specify its log id in decimal, mnemonic, or hex value. Specific CDW10/11/12/13/15 options and general options can be specified or toggled on by specifying their name and, optionally, the value wanted. When using multiple options for a single get log, separate them with a “.”(period). These get logs are similar to their `nvme cli` equivalent with exception of some slightly differing values. To view the get logs, look in the .log file generated after running the command or in the output displayed in the console if running from the command line.

Example: "`pain --nvme-get-log=error.rae.numd:10,cse.uuid:1 -f/dev/nvme0n1`"

```
 NVMe Get Log Page: LID=1 (0x01) Error Information, LSP=0 (0x00), RAE=1 (0x01), LSID=0 (0x0000), NUMD=10
(0x0000000A), LPO=0 (0x0000000000000000), uuidIndex=0 (0x00)
        0 valid error information entries

    NVMe Get Log Page: LID=5 (0x05) Commands Supported And Effects, LSP=0 (0x00), RAE=0 (0x00), LSID=0
(0x0000), NUMD=1023 (0x000003FF), LPO=0 (0x0000000000000000), uuidIndex=1 (0x01)
        ADM 0 (0x00) Delete I/O Submission Queue: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 1 (0x01) Create I/O Submission Queue: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 2 (0x02) Get Log Page: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 4 (0x04) Delete I/O Completion Queue: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 5 (0x05) Create I/O Completion Queue: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 6 (0x06) Identify: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 8 (0x08) Abort: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 9 (0x09) Set Features: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 10 (0x0A) Get Features: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 12 (0x0C) Asynchronous Event Request: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 16 (0x10) Firmware Commit: 17 (0x00000011): CSUPP, CCC, CSE 0 (no restriction)
        ADM 17 (0x11) Firmware Image Download: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 20 (0x14) Device Self-test: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 128 (0x80) Format NVM: 65543 (0x00010007): CSUPP, LBCC, NCC, CSE 1 (restriction on same
namespace)
      ADM 129 (0x81) Security Send: 131075 (0x00020003): CSUPP, LBCC, CSE 2 (restriction on any namespace)
        ADM 130 (0x82) Security Receive: 131075 (0x00020003): CSUPP, LBCC, CSE 2 (restriction on any
namespace)
        ADM 132 (0x84) Sanitize: 131075 (0x00020003): CSUPP, LBCC, CSE 2 (restriction on any namespace)
        ADM 192 (0xC0) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 226 (0xE2) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 228 (0xE4) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 229 (0xE5) Vendor specific: 3 (0x00000003): CSUPP, LBCC, CSE 0 (no restriction)
        ADM 230 (0xE6) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 233 (0xE9) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 234 (0xEA) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        ADM 250 (0xFA) Vendor specific: 1 (0x00000001): CSUPP, CSE 0 (no restriction)

        NVM 0 (0x00) Flush: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        NVM 1 (0x01) Write: 3 (0x00000003): CSUPP, LBCC, CSE 0 (no restriction)
        NVM 2 (0x02) Read: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        NVM 4 (0x04) Write Uncorrectable: 3 (0x00000003): CSUPP, LBCC, CSE 0 (no restriction)
        NVM 5 (0x05) Compare: 1 (0x00000001): CSUPP, CSE 0 (no restriction)
        NVM 8 (0x08) Write Zeroes: 3 (0x00000003): CSUPP, LBCC, CSE 0 (no restriction)
        NVM 9 (0x09) Dataset Management: 65539 (0x00010003): CSUPP, LBCC, CSE 1 (restriction on same
namespace)
```

Example: `pain --nvme-get-log=smart.dumpraw -f/dev/nvme0n1`

Since dumpraw was specified for the SMART get log, which has a log id of 2, a .bin file will be generated in the working directory of where the command was run. The naming convention for the file is `lid_<log id>_<device name>.bin`. For this example the file was named `lid_2_nvme0n1.bin.`

```
NVMe Get Log Page: LID=2 (0x02) SMART / Health Information, LSP=0 (0x00), RAE=0 (0x00), LSID=0 (0x0000),
NUMD=127 (0x0000007F), LPO=0 (0x0000000000000000), uuidIndex=0 (0x00)
        Critical Warning: 0 (0x00)
        Composite Temperature: 315 (0x013B)
        Available Spare: 100 (0x64)
        Available Spare Threshold: 10 (0x0A)
        Percentage Used: 0 (0x00)
        Endurance Group Critical Warning Summary: 0 (0x00)
        Data Units Read: 15711052 (0x00000000000000000000000000EFBB4C)
        Data Units Written: 13512723 (0x00000000000000000000000000CE3013)
        Host Read Commands: 152011717 (0x0000000000000000000000000090F83C5)
        Host Write Commands: 105618180 (0x00000000000000000000000000064B9B04)
        Controller Busy Time: 164 (0x000000000000000000000000000000A4)
        Power Cycles: 11 (0x0000000000000000000000000000000B)
Power On Hours: 1065 (0x00000000000000000000000000000429)
Unsafe Shutdowns: 6 (0x00000000000000000000000000000006)
Media and Data Integrity Errors: 2 (0x00000000000000000000000000000002)
Number of Error Information Log Entries: 2 (0x00000000000000000000000000000002)
Warning Composite Temperature Time: 0 (0x00000000)
Critical Composite Temperature Time: 0 (0x00000000)
Temperature Sensor 1: 0 (0x00000000)
Temperature Sensor 2: 0 (0x00000000)
Temperature Sensor 3: 0 (0x00000000)
Temperature Sensor 4: 0 (0x00000000)
Temperature Sensor 5: 0 (0x00000000)
Temperature Sensor 6: 0 (0x00000000)
Temperature Sensor 7: 0 (0x00000000)
Temperature Sensor 8: 0 (0x00000000)
Thermal Management Temperature 1 Transition Count: 0 (0x00000000)
Thermal Management Temperature 2 Transition Count: 0 (0x00000000)
Total Time For Thermal Management Temperature 1: 0 (0x00000000)
Total Time For Thermal Management Temperature 2: 0 (0x00000000)
```

## SMART Log

MLTT supports the retrieval of the SMART logs by specifying the MLTT switch `--smart` to either an ATA or NVMe drive. The .log file generated at the end of the test contains the retrieved SMART attributes and data in CSV format for ATA devices or in a list of fields for NVMe devices. The following examples are the outputs of .log files for each device supported.

For an ATA device:

```
pain --smart -f\\.\physicaldrive1
```

```
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` | The SMART attribute ID (the hexadecimal equivalent in shown in parentheses). |
| `VALUE` | 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)` | The 16-bit flags for the attribute retrieved from the drive (the hexadecimal equivalent in shown in parentheses). |
| `VENDOR SPECIFC DATA` | 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 |

For a NVMe device:

```
pain --smart -f/dev/nvme0n1
```

```
SMART: Target 1: "/dev/nvme0n1" - OK
Critical Warning: 0 (0x00)
    Composite Temperature: 315 (0x013B)
    Available Spare: 100 (0x64)
    Available Spare Threshold: 10 (0x0A)
    Percentage Used: 0 (0x00)
    Endurance Group Critical Warning Summary: 0 (0x00)
    Data Units Read: 15715065 (0x00000000000000000000000000EFCAF9)
    Data Units Written: 13515975 (0x00000000000000000000000000CE3CC7)
    Host Read Commands: 152054646 (0x00000000000000000000000009102B76)
    Host Write Commands: 105643588 (0x0000000000000000000000000064BFE44)
    Controller Busy Time: 164 (0x000000000000000000000000000000A4)
    Power Cycles: 11 (0x0000000000000000000000000000000B)
    Power On Hours: 1066 (0x0000000000000000000000000000042A)
    Unsafe Shutdowns: 6 (0x00000000000000000000000000000006)
    Media and Data Integrity Errors: 2 (0x00000000000000000000000000000002)
    Number of Error Information Log Entries: 2 (0x00000000000000000000000000000002)
    Warning Composite Temperature Time: 0 (0x00000000)
    Critical Composite Temperature Time: 0 (0x00000000)
    Temperature Sensor 1: 0 (0x00000000)
    Temperature Sensor 2: 0 (0x00000000)
    Temperature Sensor 3: 0 (0x00000000)
    Temperature Sensor 4: 0 (0x00000000)
    Temperature Sensor 5: 0 (0x00000000)
    Temperature Sensor 6: 0 (0x00000000)
    Temperature Sensor 7: 0 (0x00000000)
    Temperature Sensor 8: 0 (0x00000000)
    Thermal Management Temperature 1 Transition Count: 0 (0x00000000)
    Thermal Management Temperature 2 Transition Count: 0 (0x00000000)
    Total Time For Thermal Management Temperature 1: 0 (0x00000000)
    Total Time For Thermal Management Temperature 2: 0 (0x00000000)
```

This output is similar to the output from the `nvme cli` command `nvme smart-log <device>`.

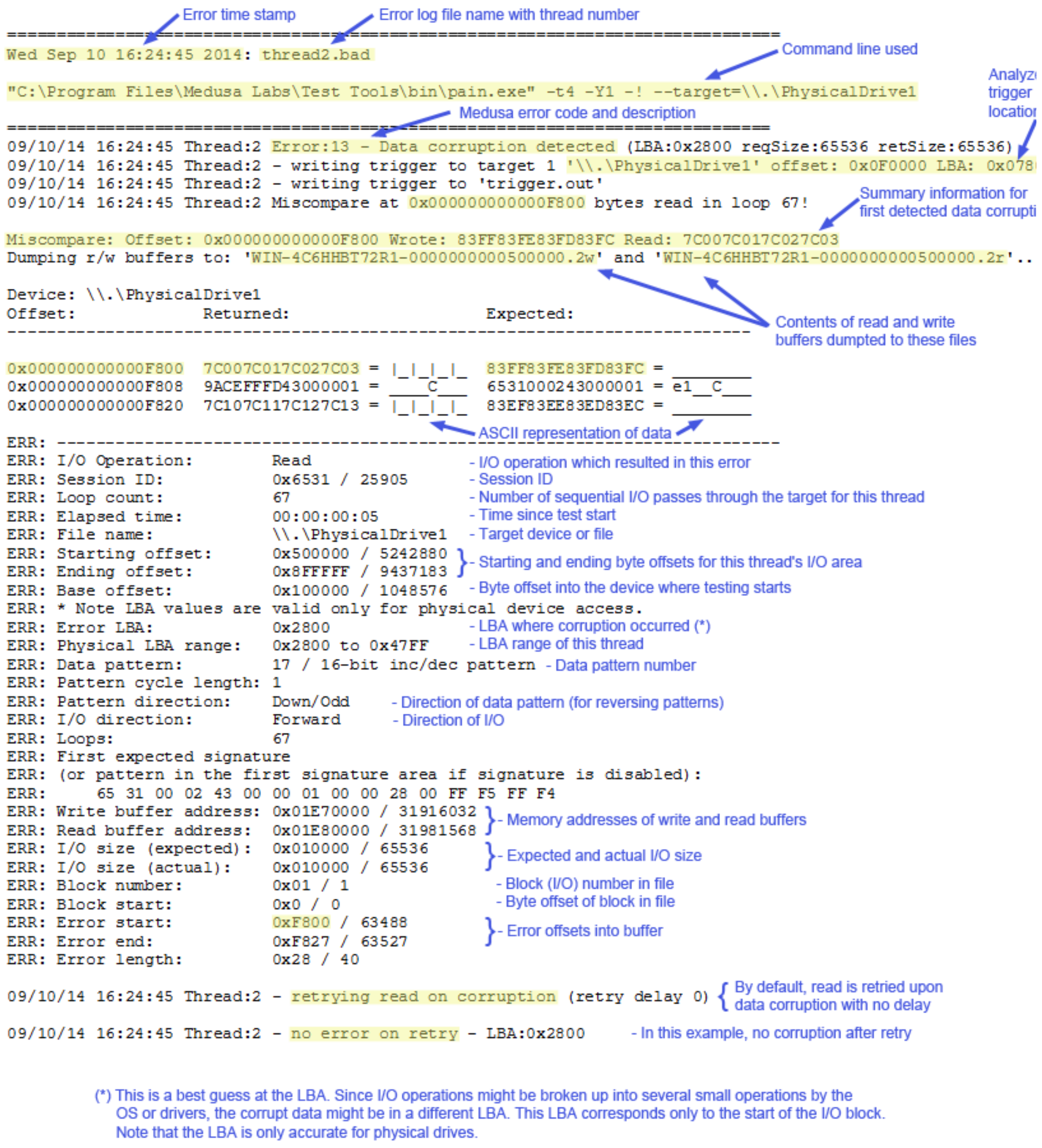
## Sample Logs

This section contains sample logs generated by MLTT.

### Sample Error Log

Figure 69 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 69** 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: 0x078
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
```

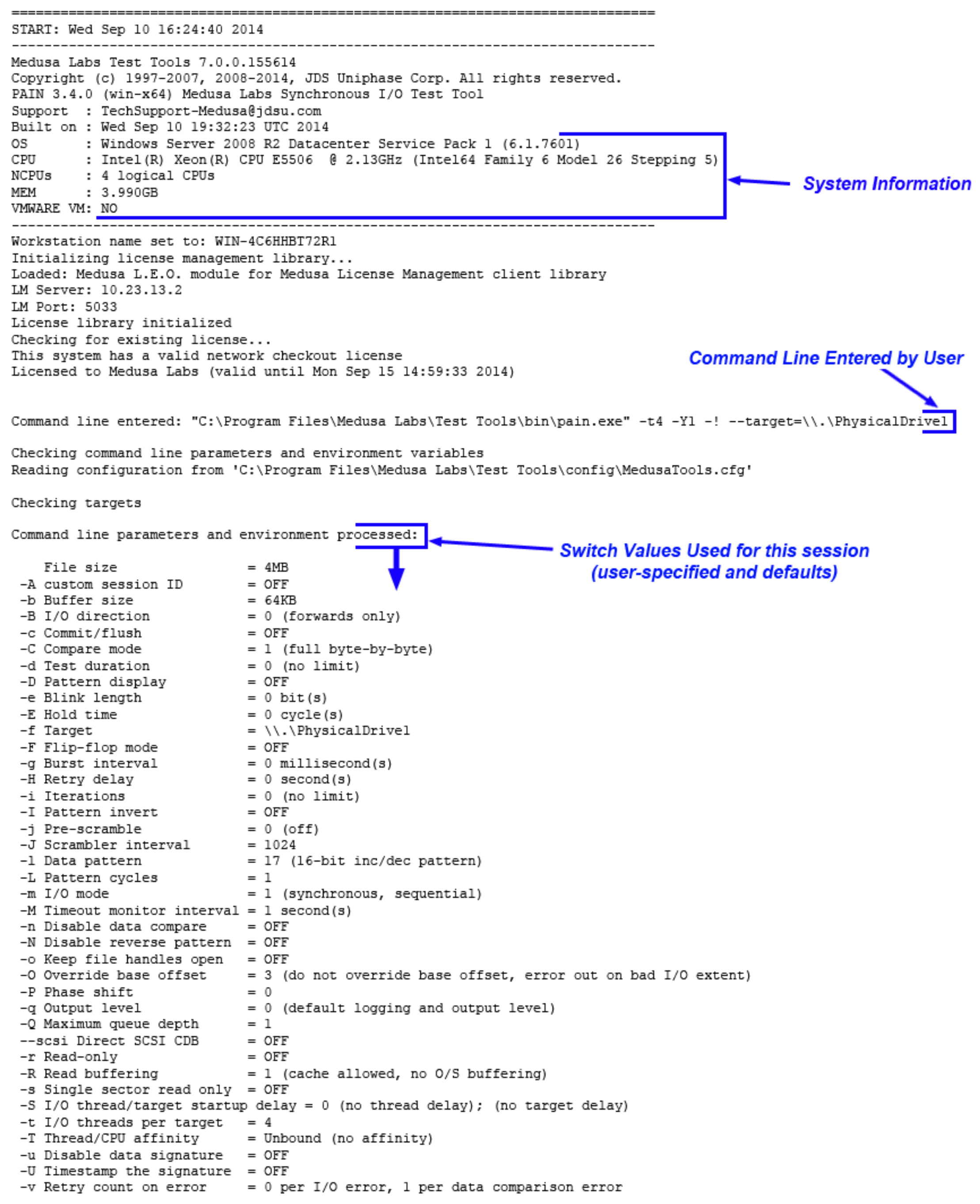
Error time stamp
Error log file name with thread number
Command line used
Analyz
trigger
locatio
Medusa error code and description
Summary information for
first detected data corrupti
Contents of read and write
buffers dumpted to these files
ASCII representation of data
- I/O operation which resulted in this error
- Session ID
- Number of sequential I/O passes through the target for this thread
- Time since test start
- Target device or file
- Starting and ending byte offsets for this thread's I/O area
- Byte offset into the device where testing starts
- LBA where corruption occurred (*)
- LBA range of this thread
- Data pattern number
- Direction of data pattern (for reversing patterns)
- Direction of I/O
- Memory addresses of write and read buffers
- Expected and actual I/O size
- Block (I/O) number in file
- Byte offset of block in file
- Error offsets into buffer
By default, read is retried upon
data corruption with no delay
- In this example, no corruption after retry

(*) 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

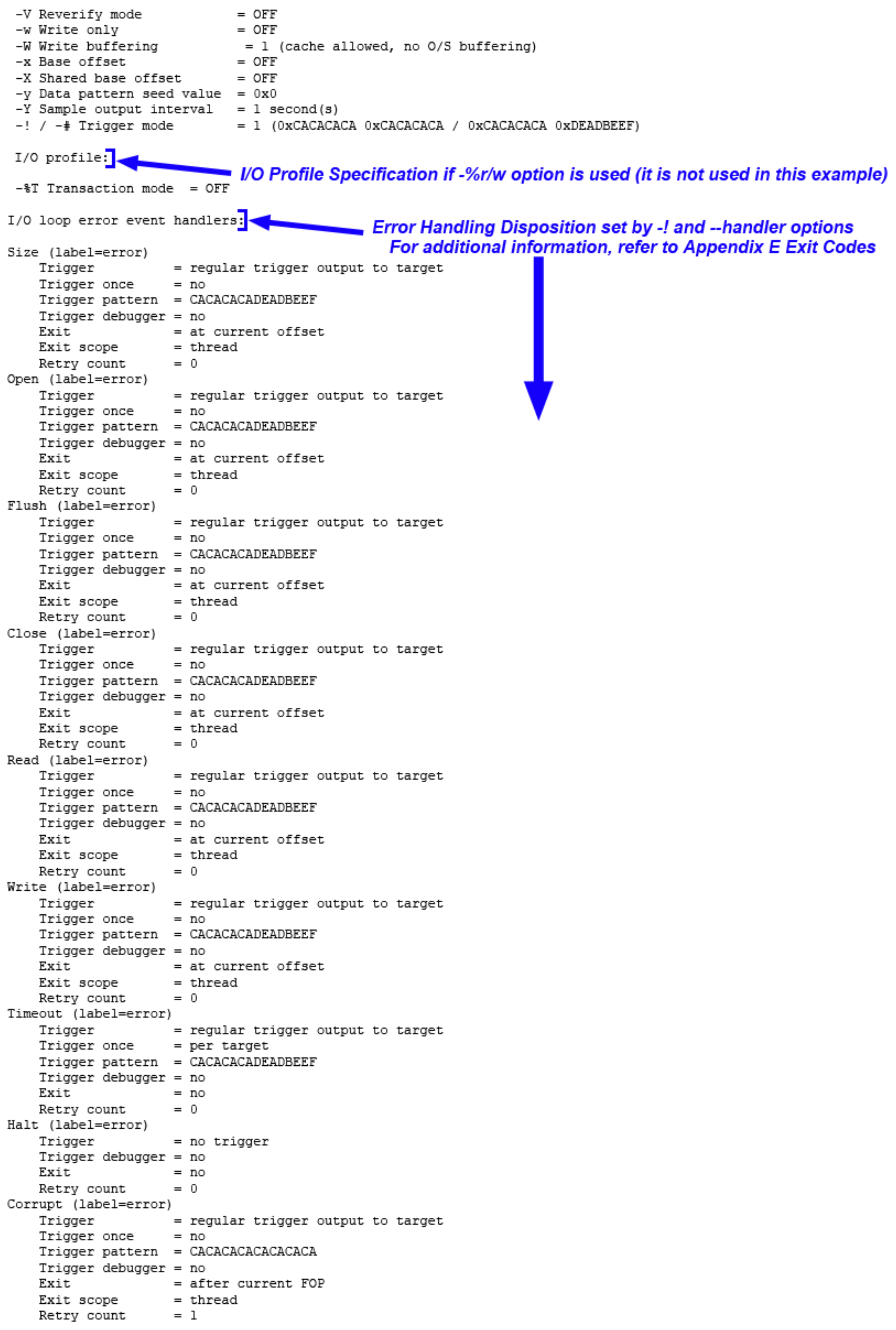
Figure 70, Figure 71, and Figure 72 show a sample status log.

**Figure 70** Status Log (Page 1 of 3)


System Information
Command Line Entered by User
Switch Values Used for this session
(user-specified and defaults)

```
================================================================================
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
```

**Figure 71** Status Log (Page 2 of 3)

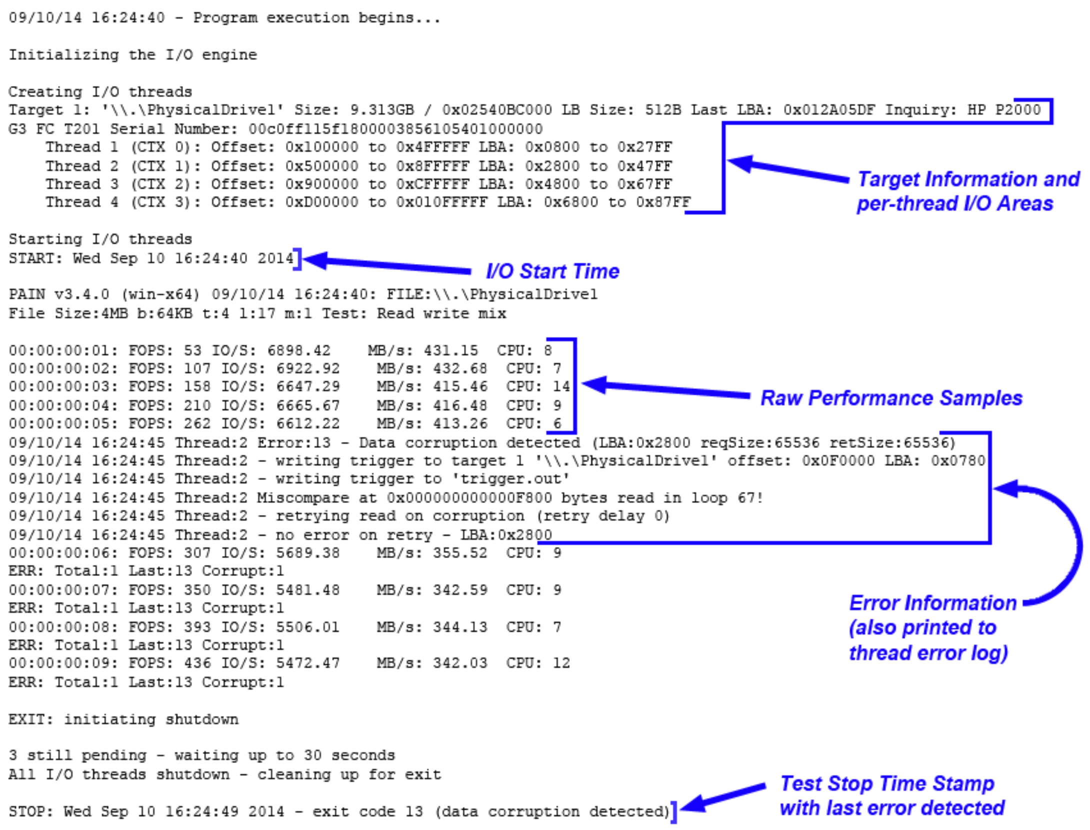


```
 -V Reverify mode            = OFF
 -w Write only               = OFF
 -W Write buffering           = 1 (cache allowed, no O/S buffering)
 -x Base offset              = OFF
 -X Shared base offset       = OFF
 -y Data pattern seed value  = 0x0
 -Y Sample output interval   = 1 second(s)
 -! / -# Trigger 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 72** Status Log (Page 3 of 3)


Target Information and per-thread I/O Areas
I/O Start Time
Raw Performance Samples
Error Information (also printed to thread error log)
Test Stop Time Stamp with last error detected

```
09/10/14 16:24:40 - Program execution begins...

Initializing the I/O engine

Creating I/O threads
Target 1: '\\.\PhysicalDrive1' Size: 9.313GB / 0x02540BC000 LB Size: 512B Last LBA: 0x012A05DF Inquiry: HP P2000
G3 FC T201 Serial Number: 00c0ff115f18000038561054010000000
    Thread 1 (CTX 0): Offset: 0x100000 to 0x4FFFFF LBA: 0x0800 to 0x27FF
    Thread 2 (CTX 1): Offset: 0x500000 to 0x8FFFFF LBA: 0x2800 to 0x47FF
    Thread 3 (CTX 2): Offset: 0x900000 to 0xCFFFFF LBA: 0x4800 to 0x67FF
    Thread 4 (CTX 3): Offset: 0xD00000 to 0x010FFFFF LBA: 0x6800 to 0x87FF

Starting I/O threads
START: Wed Sep 10 16:24:40 2014

PAIN v3.4.0 (win-x64) 09/10/14 16:24:40: FILE:\\.\PhysicalDrive1
File Size:4MB b:64KB t:4 l:17 m:1 Test: Read write mix

00:00:00:01: FOPS: 53 IO/S: 6898.42    MB/s: 431.15  CPU: 8
00:00:00:02: FOPS: 107 IO/S: 6922.92    MB/s: 432.68  CPU: 7
00:00:00:03: FOPS: 158 IO/S: 6647.29    MB/s: 415.46  CPU: 14
00:00:00:04: FOPS: 210 IO/S: 6665.67    MB/s: 416.48  CPU: 9
00:00:00:05: FOPS: 262 IO/S: 6612.22    MB/s: 413.26  CPU: 6
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!
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
00:00:00:06: FOPS: 307 IO/S: 5689.38    MB/s: 355.52  CPU: 9
ERR: Total:1 Last:13 Corrupt:1
00:00:00:07: FOPS: 350 IO/S: 5481.48    MB/s: 342.59  CPU: 9
ERR: Total:1 Last:13 Corrupt:1
00:00:00:08: FOPS: 393 IO/S: 5506.01    MB/s: 344.13  CPU: 7
ERR: Total:1 Last:13 Corrupt:1
00:00:00:09: FOPS: 436 IO/S: 5472.47    MB/s: 342.03  CPU: 12
ERR: Total:1 Last:13 Corrupt:1

EXIT: initiating shutdown

3 still pending - waiting up to 30 seconds
All I/O threads shutdown - cleaning up for exit

STOP: Wed Sep 10 16:24:49 2014 - exit code 13 (data corruption detected)
```

6

# Data Pattern Reference

This chapter describes the data patterns and their use. Topics discussed in this chapter are as follows:

## 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.

### Static Data Patterns versus Dynamic Data Patterns

Most MLTT data patterns are static which means that data patterns are generated once into the write buffer(s) of each I/O thread during initialization. Throughout testing, the same buffer(s) – therefore, same payload content that was generated once at the beginning – is written out to the target.

Data signatures embedded at the beginning of every logical block does add a little bit of dynamic variability per each write as signatures are updated every write. Even the random" data pattern, "-I35", is static, meaning the data repeats every "queue_depth x buffer_size" bytes.

However, a few data patterns are dynamic; for example, "-I32", "-I33", "-I80". For these data patterns, the write buffer(s) are filled with new content before each write. For these data patterns, application-level latency is introduced between each write.

## 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. A "reverse" pattern is generally defined as byte-by-byte reversal in order of the normal "forward" pattern. Basically, it's another pattern based on the selected pattern. This only impacts sequential-access mode. For every sequential FOP, pain/maim alternates between forward and reverse pattern buffers to write out to the target. I.e. write the forward pattern for first FOP; write the reverse pattern for next FOP; write the forward pattern for the next FOP; etc.

   It can be explicitly disabled using the "`-N`" as described in "-N Disable Data Pattern Reversals" on page 228. Note that it also has an impact on the overall buffer space allocation (see "Memory Utilization" on page 13.

   During a test, the data pattern is written out "forward" in one write pass and "reversed" in the next pass, continuously.

Example (pattern number `-l11`):

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 K "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" on page 307
  - "-L Number of Times to Repeat the Data Pattern Cycle" on page 307
  - "-I Invert Pattern Mode" on page 308
  - "-P Modify Data Patterns with a Phase Shift" on page 308
- "Custom Blink Pattern (-l99)" on page 311
  - "-e Length of Blinking Bits" on page 312
  - "-E Set Custom Blink Hold Cycles (before transition for bit-walking variations)" on page 313
  - "-F Reset custom blinking pattern to the initial pattern value each cycle" on page 313
- "Deduplication/Compression Pattern (-l80)" on page 314
  - "-E Entropy Strength" on page 316
  - "-L Deduplication Percentage" on page 317
  - "-@ Deduplication Unit" on page 317
  - "-e Duplicate Block Count" on page 318
  - "-y Random Seed Value" on page 318

## Using Pattern Modifiers

### -L Number of Times to Repeat the Data Pattern Cycle

**NOTE**

This switch is used with multiple patterns. The description provided here provides information related to all data patterns except the `-l0` and `-l80` patterns.

***Usage:***

`-L`*cycle_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.

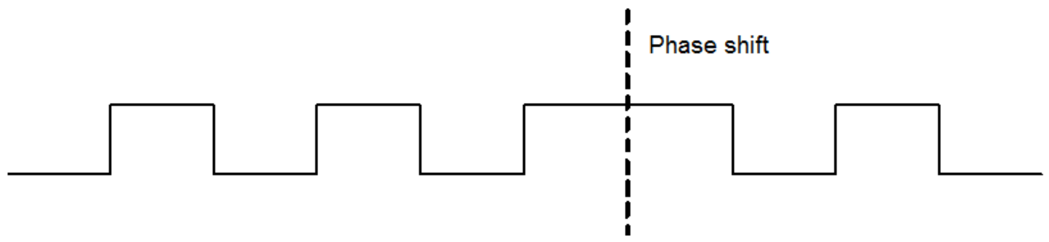
32-bit pattern: `pain -l10 -I`

```
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 select data patterns of a bus blinking nature. The effect is one of shifting the data pattern “out of phase” at the specified unit interval such that the square wave created by the blinking on/off bits reverses. See Figure 73. This is accomplished by holding the last unit value at the specified shift point one additional time before transitioning – i.e. a value of `0x00` would repeat once before transitioning to `0xFF`. Refer to the MLTT command-line help for a listing of data patterns that support this option.

**Figure 73** Data Pattern Phase Shift



The unit size varies by pattern and can be viewed by using the pattern preview switch. The unit size is indicated in bold in the command output below.

***Example:***

```
pain -l14 -D*
...
Previewing pattern '14': Noise pattern #7 - 64-bit blinking bits

Properties:

                            Unit size: 8 bytes (64-bits)
                               Period: 2 units (16 bytes)
                          Phase shift: allowed
                              Reverse: allowed
                               Invert: not allowed

Generation parameters:

                   Cycle repeat (-L): 1 units
                    Phase shift (-P): not specified
                         Invert (-I): not applicable
                Disable reversal (-N): no
                  Scramble mode (-j): none
                 Preview length (-b): 256 bytes

Forward buffer:

0000: 0000000000000000 FFFFFFFFFFFFFFFF 0000000000000000 FFFFFFFFFFFFFFFF
0020: 0000000000000000 FFFFFFFFFFFFFFFF 0000000000000000 FFFFFFFFFFFFFFFF
```

You can specify the frequency of the phase shift as the number of units to run before reversing or a default value is used if you enter –P only.

Using the `–l14` pattern as an example the base pattern is as follows:

```
pain –l14

0x0000000000000000
0xFFFFFFFFFFFFFFFF
0x0000000000000000
0xFFFFFFFFFFFFFFFF
0x0000000000000000
0xFFFFFFFFFFFFFFFF
0x0000000000000000
0xFFFFFFFFFFFFFFFF
...
```

The `–P` switch may be used to shift the pattern out of phase as shown below.

`pain –l14 –P4` **(shift every 4 units)**

| Pattern | Units |
|---|---|
| 0x0000000000000000 | **1** |
| 0xFFFFFFFFFFFFFFFF | **2** |
| 0x0000000000000000 | **3** |
| 0xFFFFFFFFFFFFFFFF | **4** |
| 0xFFFFFFFFFFFFFFFF | **(Shift)** |
| 0x0000000000000000 | **1** |
| 0xFFFFFFFFFFFFFFFF | **2** |
| 0x0000000000000000 | **3** |
| 0xFFFFFFFFFFFFFFFF | **4** |
| 0xFFFFFFFFFFFFFFFF | **(Shift)** |
| 0x0000000000000000 | **1** |
| 0xFFFFFFFFFFFFFFFF | **2** |
| ... | |

`pain –l14 –P3` **(shift every 3 units)**

| Pattern | Units |
|---|---|
| 0x0000000000000000 | **1** |
| 0xFFFFFFFFFFFFFFFF | **2** |
| 0x0000000000000000 | **3** |
| 0x0000000000000000 | **(Shift)** |
| 0xFFFFFFFFFFFFFFFF | **1** |
| 0x0000000000000000 | **2** |
| 0xFFFFFFFFFFFFFFFF | **3** |
| 0xFFFFFFFFFFFFFFFF | **(Shift)** |
| 0x0000000000000000 | **1** |
| 0xFFFFFFFFFFFFFFFF | **2** |
| ... | |

## Custom Blink Pattern (-l99)

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 Length of Blinking Bits

**NOTE**
This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l99` Custom Blink Pattern.

***Usage:***

-e*bit_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 Set Custom Blink Hold Cycles (before transition for bit-walking variations)

**NOTE**
This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l99` Custom Blink Pattern.

***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 Reset custom blinking pattern to the initial pattern value each cycle

***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.
```

## Deduplication/Compression Pattern (-l80)

The custom deduplication/compression pattern, indicated by `-l80`, is fundamentally a sequence of random bytes with adjustable parameters to test the target device's compres-sion
function or deduplication (dedup) function, or both functions together.

- When `-E` option value is set to a number less than 100, `-l80` outputs a sequence of random values mixed with strips of consecutive 0s that the target device can compress.
- When `-L` options value is set to a number greater than `0`, `-l80` outputs a sequence of random values with a mix of duplicate blocks inserted in the output which the target device can deduplicate.

The default state of `-l80`, with `-E100` and `-L0`, makes each output of the `-l80` pattern extremely difficult to compress and deduplicate. As with the other random patterns, use `-y` to set the seed to change the random value sequence.

### Compression-only Testing

With the `-l80` pattern, the keys to testing only the compression function of the target device is to adjust how much of the output data can be compressed with `-E` option while deduplicaton is prevented with the `-L0` setting. Due to the performance-versus-resource trade-off of the `-l80` pattern algorithm, there is no guarantee that the generated sequence of random values are not duplicated in the output data over time.

- If the target device does not have the deduplication function enabled, then the potential occurrences of uncontrolled duplicate blocks do not interfere with compression-only testing.
- If the target device has the deduplication function enabled, then steps must be taken to minimize unintended deduplication during I/O.

To minimize unintended deduplication, do not disable the unique data signature that MLTT embeds in the output data by default. Furthermore, use `-Um` to append a timestamp to the data signature. The best option to test only the compression function is to disable the target device's deduplication function if possible.

## Deduplication-only Testing

With the `-l80` pattern, use `-L` to specify how much of the output data can be deduplicated while compression is prevented with the `-E0` setting. The `-@` option must be set to match the target system's deduplication unit size. When `-L` is greater than `0`, the `-l80` generator keeps a pool of blocks, each one the size of deduplication unit size. Each block in this pool is filled with a unique sequence of random values. Each time `-l80` needs to insert a duplicate block in the output data, one of the blocks from this pool is duplicated into the output buffer.

Use `-e` to set the number of blocks in this pool. The number of blocks is 1, by default. This number can be increased in order to add complexity to the data. Increasing the number of blocks increases the amount of unique data segments that are used as duplicated blocks. Increasing the `-e` value makes it more difficult for a deduplication engine to track duplicated data.

The possibility of unintended deduplication still exists (as explained in <Link>“Compression-only Testing”.) To make sure that only the blocks drawn from the duplicate block set are subject to deduplication and to minimize unintended deduplication while keep the testing more deterministic (i.e. keep the amount of deduplication close to the `-L` setting,) do not disable the unique data signature that MLTT embeds in the output data by default. Furthermore, use -Um to append a timestamp to the data signature. To make sure the blocks drawn from the duplicate pool can be recognized as duplicates, they are never marked with data
signature.

If the target device also has compression function enabled, then keeping the final deduplication amount close to specified by `-L` value becomes very difficult because, even at `-E100`, `-l80` cannot generate an output sequence that is absolutely uncompressible.

Dealing with this scenario is dependent on the specific operation detail of the target device. For example, in a scenario where compression occurs before deduplication, if a duplicate block is reduced in size (even by 1 byte), there is no longer a guarantee that block will be recognized as a duplicate. Offsetting this compression requires increasing the `-@` value using a trial-and-error method. The best option to test only the deduplication function is to disable data compression in the I/O path to the target device if possible.

## Deduplication and Compression Testing

It is possible to generate the output data with both `-E` and `-L` settings applied such that the resulting output is both compressible and subject to deduplication. How deterministic such testing can be made is dependent on the specific operation details of the target device. The same concerns described in “Deduplication-only Testing” on page 315 when compression function is enabled also apply here, where testing of both functions together is desired.

In a scenario where compression occurs before deduplication, the goal is to create a data output which, after the desired amount of compression set by `-E` is applied, the resulting compressed data can be deduplicated by the desired `-L` setting amount. In theory, the effective dedup unit size must be increased to offset the compression amount. For example, if the target device has a dedup unit size of 8KB, and if `-E25` is used (i.e. compress to 25% of the original size), then a reasonable starting effective `-@` setting is 8K / 0.25 = 32KB. That is, use a proportionally larger duplicate block so that, after compression, the resulting compressed block becomes closer in size to the actual target device's dedup unit size. Nevertheless, further tuning using a trial-and-error method is required because `-E` cannot precisely control the actual compression amount.

## -E Entropy Strength

**NOTE**
This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l80` Deduplication/Compression Pattern.

***Usage:***

```
-E<entropy_strength>
```

This switch specifies how compressible the payload is.
0 (no entropy, most compressible) to 100 (most entropy, least compressible).
It basically defines the percentage of original data that is written after compression is applied.
The default value is `-E100.`

***Example:***

```
pain -l80 -E25
```

With this example, the output data should compress to about 25% of the original size.

## -L Deduplication Percentage

**NOTE**
This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l80` Deduplication/Compression Pattern.

***Usage:***

```
-L<dedup_%>
```

This switch specifies how much of the original data can be deduplicated. The percentage of duplicated data blocks can be specified from 0 to 100 percent. The default value is `-L0`.

***Example:***

```
pain -l80 -L30
```

With this example, about 70% of generated data will be unique and about 30% will be filled from a pool of duplicate blocks that can repeated (thus be deduplicated by the target device.)

## -@ Deduplication Unit

**NOTE**
This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l80` Deduplication/Compression Pattern.

***Usage:***

```
-@<dedup_unit>
```

This switch specifies the dedup block size. It should be set to whatever the value the target storage device's deduplication system uses. The default for MLTT is 8K. Note that you can use a size suffix like 'k', 'b', etc. as usual, but it will not accept the "u" suffix. "k" is the default suffix (i.e. `-@32` is `-@32K`). This option is ignored for `-L0`.

## -e Duplicate Block Count

**NOTE**

This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l80` Deduplication/Compression Pattern.

***Usage:***

```
-e<duplicate_block_count>
```

This switch sets the number of unique blocks in the duplicate block pool to draw from. Using the `-L30` example, about 30% of the data will be duplicated from a block in this pool. From the second time a block from this pool is written out, it can be deduplicated. This option is ignored for `-L0`. The default is `-e1`.

## -y Random Seed Value

**NOTE**

This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l80` Deduplication/Compression Pattern.

***Usage:***

`-y`<*random_seed_value*>

This option is used to specify a 32-bit random number seed value to the `-l80` Deduplication/Compression Pattern.

## 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 hex 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

**NOTE**

This switch is used with multiple patterns. The description provided here provides information related specifically to the `-l0` special data pattern switch.

***Usage:***

-@<path/file_name>

The `-l0` data pattern is a special data pattern number that identifies that the data pattern is to be read in from an existing file. Use the `-@` switch to identify the path and file name that contains the data pattern that will be used. The file must be greater than or equal in size to 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.

In cases of testing on remote system, a local copy of the pattern file must already exist in the path specified by "-@<path/filename>". Additionally, note that the Windows and Linux operating systems use different slash characters in their paths ("\" vs. "/"), so when testing to remote systems, Window and Linux systems will need separate commands with the appropriate slash characters in the "-@<path/filename>".

***Example:***

```
pain -b512k -l0 -@c:\data\pattern.dat
```

# Catapult Test Tool Automation

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.

Topics discussed in this chapter are as follows:

## 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

returns a list of all MLTT test stations on subnet

catapult -r<hostname or IP address>

used with `-f`,`- l`, or `-p` returns a list of targets connected to the remote system

### Drive Listing Examples

This section shows the following three drive listing examples:

- A listing of physical drives discovered on a Windows system
- A listing of physical drives discovered on a Linux system
- A file system drive discovery on a Windows system

### *Catapult Physical Drive Discovery on Windows Example*

**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 NameSCSI ID  INQUIRY DATAExclusions
-----------------------------------------------------------------
-----
  \\.\PhysicalDrive0  0:0:0:0  ST330630A               3.21AP
  \\.\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
```

### *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
```

### *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/MountExclusions
---------------------------------------------------------
     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.

**CAUTION**

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.

**CAUTION**

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 327.

**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" on page 328
- "-b Log retrieval" on page 328
- "-c Clean Directories" on page 329
- "-d Delay Test Start" on page 330
- "-e Increment Data Pattern" on page 330
- "-f File system access" on page 331
- "-g Change directory prefix" on page 332
- "-h online help" on page 333
- "-i Include drive" on page 333 (also -i.str_Include drive based on inquiry data)
- "-j Prescramble write output data" on page 334
- "-J Limit inquiry ioctls" on page 335
- "-k Kill tool processes" on page 335
- "-l Logical drive access" on page 336
- "-m Minimize tool windows" on page 337
- "-n Enable prompts" on page 338
- "-now Run all tests with no windows" on page 338
- "-o Override device exclusions" on page 339
- "--off Offline disk" on page 340
- "--on Online disk" on page 341
- "-p Physical drive access" on page 341
- "-q Removes excluded drives" on page 342
- "-r remote access" on page 343
- "--restart-service Restarts the Medusa agent" on page 344
- "-s Set tool starting offset" on page 345
- "-t Multi-target mode" on page 346
- "-v Verify mode" on page 347
- "-w Watch mode" on page 348
- "-x Exclude drive" on page 349 (also -x.str_Exclude drive based on inquire data)
- "-y Specify grace period" on page 350
- "-z Debug mode" on page 351

## -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. When using multi-target mode (catapult -t), this will not delay between startups within the multi-target tests themselves, but will pause between tests if multiple hosts are specified.

***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.

The only non-destructive mode for the tools, this will run your specified tool to a file named targetfile.dat in the topmost directory of the target you specify. Note that if there is already a file named targetfile.dat, it will be overwritten.

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

**NOTE**
This switch may only be ran to a system that has a file system other than the OS file system.

***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
```

Multiple hosts definitions can be specified by separating them with a semi-colon (for example: -ihost1:2,4,5;host2:a-d,f). If no host is given, the list will be applied to all included hosts.

**NOTE**

If you are running with a network of mixed systems (linux/windows), including only devices by letters will exclude ALL linux devices. Make sure you know which devices you are including and excluding. Inclusion does not override the tools normal exclusion options.

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 Prescramble write output data

***Usage:***

`-j`*<mode_number>*

***Description:***

Use the `-j` option to prescramble the write output data. The available mode numbers are:

| | |
|---|---|
| 0 | No pre-scramble (default) |
| 1 | SAS scramble |
| 2 | SATA scramble |

***Default:***

The timeout is `-j0`.

## -J Limit inquiry ioctls

***Usage:***

`-J`<*interval*>

***Description:***

Set the data scramble reset 'interval' to specified number of bytes if '`-j1`' or '`-j2`' is specified.

***Default:***

If used in conjunction with `-j1` then `-J1024` is the default value; if used in conjunction with `-j2`, then `-J8192` is the default value.

## -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.

Using `catapult -k -r` without specifying a host will stop all tests on all systems on the subnet.

***Example:***

```
catapult -k
catapult -k -r<hostname or IP address>
```

***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.

**CAUTION**

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' multi-target option.

***Example:***

The following example will run the tests without displaying them in the console window.

```
catapult -now -p
```

***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[*option*]

***Description:***

Allows 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. Note that the system exclusion can not be overridden.

**CAUTION**

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.

Here is the list of the exclusion options that can be overridden selectively. Devices may be excluded with the following exclusion options:

| | |
|---|---|
| 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 (physical disks only). Also, a file system found on device without partition (Linux 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. |
| Z | Device or path to device is either inaccessible or read-only. |

***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 --off
```

The following example sets the 2nd and 3rd physical drives to **Offline**.

```
    catapult -p -i2,3 --off
```

## --on Online disk

***Usage:***

```
--on <-p>
```

***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 --on
```

The following example sets the 2nd and 3rd physical drives to Online.

```
catapult -p -i2,3 --on
```

## -p Physical drive access

***Usage:***

```
-p
```

***Description:***

Use the -p switch to have Catapult scan for physical drives.

**CAUTION**

Physical drive access is destructive! Any existing data on the drive WILL be destroyed, including partition headers and volume boot records. Physical drive access should not be run on disks whose partitions you wish to keep.

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`[*host*]

***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` does not create a host.dat file. This file must be created by the user, and it also needs to be in the current working directory. 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. Putting in extra information or not using ASCII text will result in that host line being ignored.

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 -r`*server_name*

The following example runs an instance of Pain to all detected remote physical drives.

`catapult -p -r`*server_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. "-x Starting Offset" on page 219 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<option>`

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.

Verification of tests can be performed using these options:

| | |
|---|---|
| `-va<##>` | Sets a minimum average MB/s for verification. |
| `-vb<##>` | Sets a minimum average IO/s for verification. |
| `-vc` | Checks tests for data corruption errors. |
| `-ve` | Checks tests for any errors not detected by other options. |
| `-vi<##>` | Sets a maximum number of IO halts to check for. |
| `-vl` | Outputs verification information to a file vlog.log. |
| `-vm<##>` | Sets a minimum MB/s for verification. |
| `-vn<##>` | Sets a minimum IO/s for verification. |
| `-vp` | Processes all tests that are found in .csv files. |
| `-vq` | Print only summary information. |
| `-vv` | Print verbose information for tests with errors. |
| `-vvv` | Print very verbose information for all tests. |

For help on any of the specific options listed above, simply set the option you need help on and the `-h` option (for example: `-h -va`).

***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 options.

## -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:***

-x[*exclude_list*]

***Description:***

-x[*exclude_list*] provides a list of devices to exclude from testing. Exclusion can either be done:

- As a single list. For example:

  ```
  -x2,4,6-11
  ```

- As a list specifying a host and targets separated by a colon. For example:

  ```
  -xlocalhost:2,4,6-11
  ```

Multiple hosts definitions can be specified by separating them with a semi-colon (for example:

```
-xhost1:2,4,5;host2:2-5,8
```

If no host is given, the list will be applied to all included hosts.

**NOTE**

If you are running with a network of mixed systems (linux/windows), excluding only devices by letters will include ALL linux devices. Make sure you know which devices you are including and excluding. Devices not listed in the -x option and not excluded by the tools normally will be the devices tested.

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. This mode is typically only used when working with Technical Support to troubleshoot target discovery 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 physical drives that have not been excluded. 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 330 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 physical drives that have not been excluded. 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 (-l99)" on page 261.

```
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
```

A

# Data Pattern Numbers

This appendix lists the data pattern numbers and the description of each data pattern as they are listed in the Pain and Maim help.

**Table 15** Data Pattern Numbers and Descriptions

| Number | Description |
|---|---|
| 0 | pattern read from file |
| 1 | Repeating 8-bit value (use '-y<value>' to specify the 8-bit pattern 'value') |
| 2 | Repeating 16-bit value (use '-y<value>' to specify the 16-bit pattern 'value') |
| 3 | 32-bit value - Fixed pattern - 1st FOP 0xFFFFFFFF, 2nd:0x00000000 |
| 4 | Repeating 32-bit value (use '-y<value>' to specify the 32-bit pattern 'value') |
| 5 | Noise pattern #1 - 32-bit checkerboard |
| 6 | Noise pattern #2 - 32-bit blinking bus |
| 7 | Noise pattern #3 - walking/XOR bits |
| 8 | 8-bit incrementing/decrementing pattern |
| 9 | 8-bit double inc/dec pattern |
| 10 | 8-bit quad inc/dec pattern |
| 11 | Noise pattern #4 - SCSI blinking bits |
| 12 | Noise pattern #5 - SCSI alternating parity tester |
| 13 | Noise pattern #6 - checkerboard |
| 14 | Noise pattern #7 - 64-bit blinking bits |
| 15 | Noise pattern #8 - 64-bit walking/XOR bits |
| 16 | 16-bit incrementing pattern |
| 17 | 16-bit inc/dec pattern |
| 18 | FC LF noise pattern #1 - alternating low freq. jitter |
| 19 | FC LF noise pattern #2 - low freq. jitter |
| 20 | FC LF noise pattern #3 - low freq. jitter |
| 21 | FC LT noise pattern #1 - alternating low transition density |
| 22 | FC LT noise pattern #2 - low transition density |
| 23 | FC LT noise pattern #3 - low transition density |
| 24 | FC HT noise pattern #1 - high transition density |
| 25 | FC CJTPAT noise pattern - all transition |
| 26 | FC CSPAT noise pattern - supply noise test |
| 27 | FC JTPAT noise pattern #1 - receive jitter tolerance |
| 28 | 64-bit Alternating Blinking Bits |
| 29 | 64-bit Incrementing bits (per byte) |
| 30 | Ethernet network noise pattern |
| 31 | 16-bit checkerboard |
| 32 | 32-bit incrementing pattern, repeats per FOP |
| 33 | 32-bit continuous increment pattern, repeats every 4GB |
| 34 | 10G continuous jitter pattern |

**Table 15** Data Pattern Numbers and Descriptions

| Number | Description |
|---|---|
| 35 | Random pattern<br>(use '-y<seed>' to specify an optional random 'seed' between '0' and '0xFFFFFFFF') |
| 36 | Chipset noise pattern #1 - modified I18 |
| 37 | FC Low Frequency Transitions |
| 38 | FC neutral noise pattern #1 |
| 39 | FC neutral noise pattern #2 |
| 40 | FC blink pattern #1 |
| 41 | 64-bit incrementing pattern |
| 42 | 16-bit alternating blink pattern |
| 43 | FC ISI killer pattern |
| 44 | FC 1KJPAT |
| 45 | Walking flip/flop bytes |
| 46 | Data pattern set to buffer memory addresses.<br>This data pattern should not be used with the data re-verification option (`-V`), since the buffer memory address are likely to have been re-assigned between test runs. |
| 47 | Scrambler pattern 1<br>(use '-y<seed>' to specify an optional random 'seed' between '0' and '0xFFFFFFFF') |
| 48 | Scrambler pattern 2 |
| 49 | Scrambler pattern 3 |
| 50 | FC blink pattern #2 |
| 51 | FC blink pattern #3 |
| 52 | FC blink pattern #4 |
| 53 | FC blink pattern #5 |
| 54 | SATA Composite-bit Pattern |
| 55 | SATA Low Transition Density Pattern |
| 56 | SATA High Transition Density Pattern |
| 57 | SATA Low Frequency Spectral Content Pattern |
| 58 | SATA Simultaneous Switching Outputs Pattern |
| 59 | SATA Lone-Bit Pattern |
| 60 | FC random neutral pattern<br>(use '-y<seed>' to specify an optional random 'seed' between '0' and '0xFFFFFFFF') |
| 61 | SAS CJTPAT |
| 62 | FC random neutral pattern with inversion<br>(use '-y<seed>' to specify an optional random 'seed' between '0' and '0xFFFFFFFF') |
| 63 | FC JSPAT |
| 64 | FC JTSPAT |
| 69 | Fixed pattern of 0 |
| 70 | Data pattern set to I/O LBA |
| 71 | Data pattern set to I/O byte offset |

**Table 15** Data Pattern Numbers and Descriptions

| **Number** | **Description** |
|---|---|
| 80 | Compressible random pattern<br>(use '-y<seed>' to specify an optional random 'seed' between '0' and '0xFFFFFFFF').<br>Use this pattern to test compression and/or dedup.<br>Use<br>'-E' for compression entropy (0 to 100, default 100),<br>'-L' for dedup % (0 to 100, default 0),<br>'-e' for number of duplicate blocks (default 1), and<br>'-@' to specify the dedup unit size (default 8KB). |
| 94 | PCI parity alternating pattern |
| 96 | 16-bit Alternating 0xAAAA, 0x5555 |
| 99 | Custom Blink<br>-l99  No walking bit<br>-l99w Walk both '1' and '0' bits<br>-l99o Walk '0' bits only<br>-l99f Walk '1' bits only<br>Use '-L' for bit length, '-e' for length of '1' bits, '-E' for blink hold cycles, and '-F' to reset each cycle |

B

# Test Guidelines and Examples

This appendix shows examples of Medusa Labs Test Tools (MLTT) used in common test scenarios.The topics discussed in this appendix are as follows:

## 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 the case of PCIe, the available data will vary with the PCIe version. PCIe 1.0 can support up to 250 MB/s (per lane), PCIe 2.0 can support up to 500 MB/s (per lane), PCIe 3.0 can support up to 984.6 MB/s (per lane), and PCIe 4.0 can support up to 1969.2 MB/s (per lane). 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 -t128 -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 -t4 -Q256 -b512 512 -u -n -m16
```

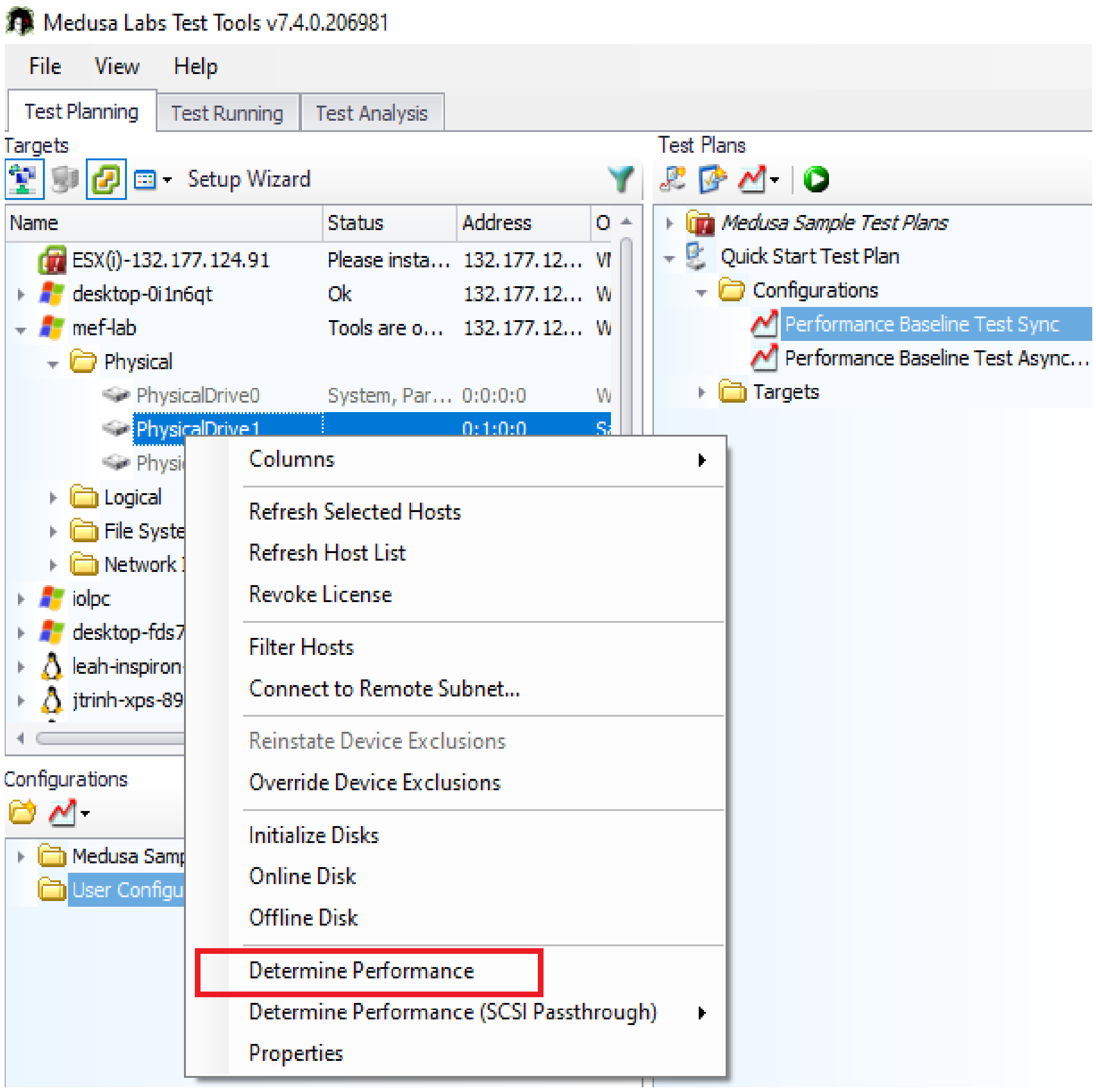
The above examples are full-duplex-style tests. It is always a good idea to run half duplex tests (write only and read only.) To do this, use the `-w` or `-r` switches with command lines similar to those given in the examples.

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.)

## General Guideline:

When performance testing, especially with NVMe drives, use asynchronous I/O (maim) and keep increasing the thread and queue depth count until maximum performance is achieved. To do this, start with one thread and increase the queue depth by 1 until performance is maximized. Then repeat the process; keeping queue depth the same and this time increasing the thread count.

Alternatively, MLTT can find the optimal performance of a drive with the “Determine Performance” tool. To use this tool, right click on a drive and select the “Determine Performance” option.



## 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" 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 K "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
```

C

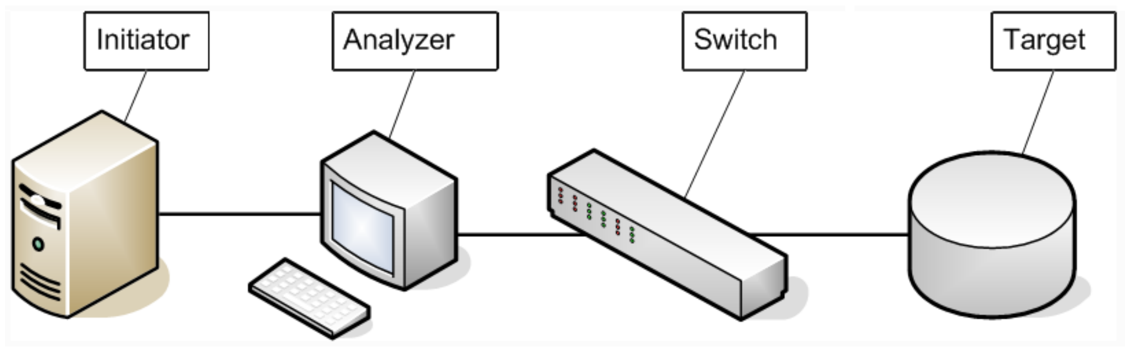
# Debug Example

This appendix contains a sample debug of a data corruption scenario. The topics discussed in this appendix are as follows:

## Overview

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 VIAVI Xgig Analyzer between the HBA and a switch.

**Figure 74** Fibre Channel Debug Test Configuration

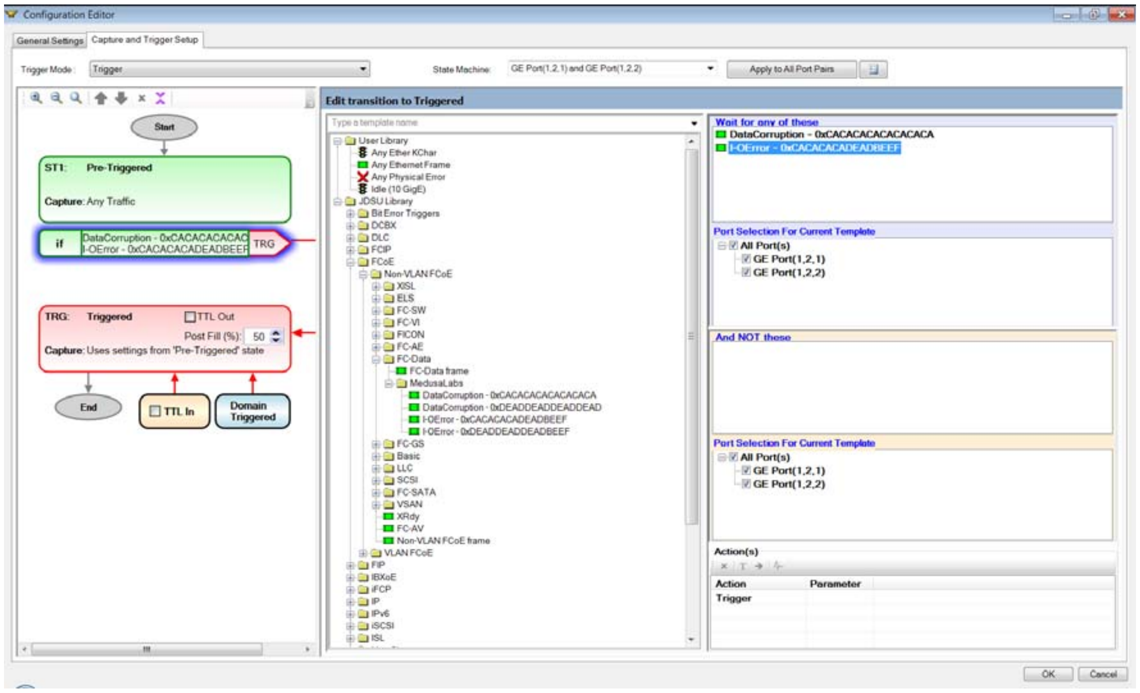


In this configuration, a basic Pain stress test is run on a Windows initiator to a physical drive on the target device. The analyzer is set to trigger on a special value sent by MLTT when the `-!` or `-#` switch is used. The trace buffer should be set to allow sufficient room for capture of relevant 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. See Figure 75.

**Figure 75** Setting the Default Trigger Value on the Analyzer



Trigger Mode: Trigger

State Machine: GE Port(1,2,1) and GE Port(1,2,2)

ST1: Pre-Triggered
Capture: Any Traffic

if DataCorruption - 0xCACACACACAC
I-OError - 0xCACACACADEADBEEF TRG

TRG: Triggered ☐ TTL Out
Post Fill (%): 50
Capture: Uses settings from 'Pre-Triggered' state

Edit transition to Triggered

MedusaLabs:
DataCorruption - 0xCACACACACACACACA
DataCorruption - 0xDEADDEADDEADDEAD
I-OError - 0xCACACACADEADBEEF
I-OError - 0xDEADDEADDEADBEEF

Wait for any of these
DataCorruption - 0xCACACACACACACACA
I-OError - 0xCACACACADEADBEEF

Port Selection For Current Template
☑ All Port(s)
☑ GE Port(1,2,1)
☑ GE Port(1,2,2)

And NOT these

Port Selection For Current Template
☑ All Port(s)
☑ GE Port(1,2,1)
☑ GE Port(1,2,2)

Action(s)

| Action | Parameter |
|---|---|
| Trigger | |

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<br><br>2 - Corrupted LBA<br>3 - Starting LBA | 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 |

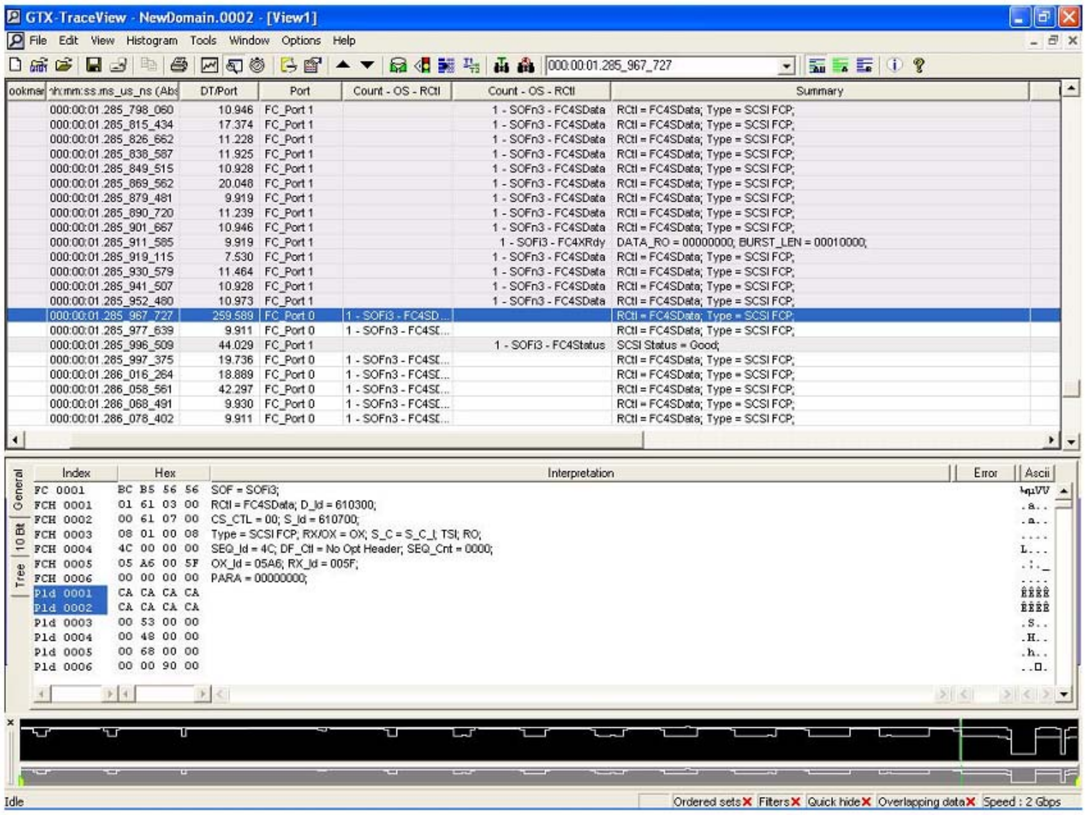
| Word | Values | Conditions |
|---|---|---|
| | 4 - Ending LBA<br>5 - Starting offset (bytes)<br>6 - Ending offset (bytes)<br>7 - Base offset (bytes) | |
| 8 | Program Name | First four characters |
| 9 | WS_NAME | Last four characters |
| 10 | Flags<br>Intel = DDCCBBAA<br>Sparc = AABBCCDD | DD = thread_num<br>CC = 01 if buffer reversed, 00 if forwards<br>BB = 00 if I/Os forward, 01 if backwards<br>AA = Data pattern (-l value) - in HEX |
| 11 | Loop count | |
| 12 | Buffer Size | |
| 13 | Completed I/Os | |
| 14 | Block# / Error Type<br>Intel = DDCCBBAA<br>Sparc = AABBCCDD | AA = Error Type (as identified in the following list)<br>`02 (2) STARTUP_ERROR    0C (12) WRITE_ERROR`<br>`03 (3) MALLOC_ERROR     0D (13) CORRUPT_ERROR`<br>`04 (4) LOG_ERROR        0E (14) INITIAL_ERROR`<br>`05 (5) SEEK_ERROR       0F (15) REMOVE_ERROR`<br>`06 (6) RETRY_ERROR      10 (16) UNKNOWN_ERROR`<br>`07 (7) SIZE_ERROR       11 (17) TIMEOUT_ERROR`<br>`08 (8) OPEN_ERROR       12 (18) LICENSE_ERROR`<br>`09 (9) FLUSH_ERROR      13 (19) IOCTL_ERROR`<br>`0A (10) CLOSE_ERROR     14 (20) HALT_ERROR`<br>`0B (11) READ_ERROR`<br>BBCCDD = Block Number |
| 15 | Index info<br>Intel = DDCCBBAA<br>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. See Figure 76.

**Figure 76** Locating the Trigger Data Frame in TraceView



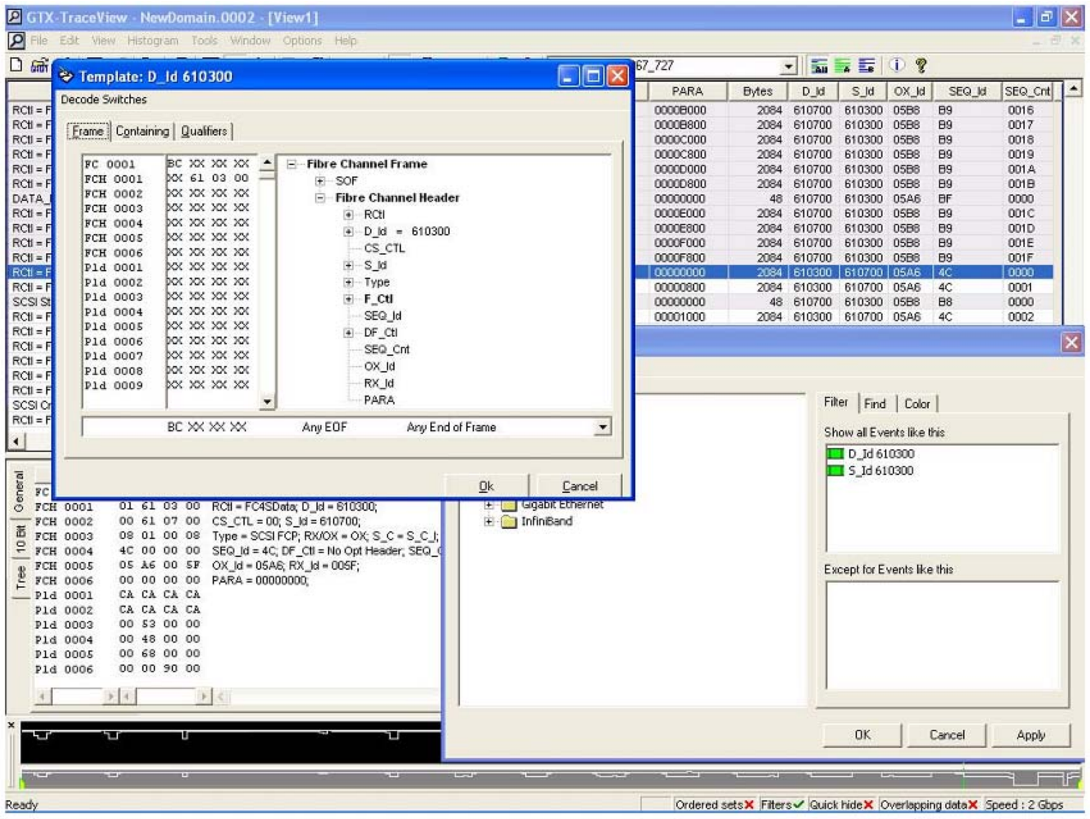
| ookmar | hh:mm:ss.ms_us_ns (Abs | DT/Port | Port | Count - OS - RCtl | Count - OS - RCtl | Summary |
|---|---|---|---|---|---|---|
| | 000:00:01.285_798_060 | 10.946 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_815_434 | 17.374 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_826_662 | 11.228 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_838_587 | 11.925 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_849_515 | 10.928 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_869_562 | 20.048 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_879_481 | 9.919 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_890_720 | 11.239 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_901_667 | 10.946 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_911_585 | 9.919 | FC_Port 1 | | 1 - SOFi3 - FC4XRdy | DATA_RO = 00000000; BURST_LEN = 00010000; |
| | 000:00:01.285_919_115 | 7.530 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_930_579 | 11.464 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_941_507 | 10.928 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_952_480 | 10.973 | FC_Port 1 | | 1 - SOFn3 - FC4SData | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_967_727 | 259.589 | FC_Port 0 | 1 - SOFi3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_977_639 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.285_996_509 | 44.029 | FC_Port 1 | | 1 - SOFi3 - FC4Status | SCSI Status = Good; |
| | 000:00:01.285_997_375 | 19.736 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.286_016_264 | 18.889 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.286_058_561 | 42.297 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.286_068_491 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| | 000:00:01.286_078_402 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |

| Index | Hex | Interpretation | Error | Ascii |
|---|---|---|---|---|
| FC 0001 | BC B5 56 56 | SOF = SOFi3; | | ¼µVV |
| FCH 0001 | 01 61 03 00 | RCtl = FC4SData; D_Id = 610300; | | .a.. |
| FCH 0002 | 00 61 07 00 | CS_CTL = 00; S_Id = 610700; | | .a.. |
| FCH 0003 | 08 01 00 08 | Type = SCSI FCP; RX/OX = OX; S_C = S_C_I; TSI; RO; | | .... |
| FCH 0004 | 4C 00 00 00 | SEQ_Id = 4C; DF_Ctl = No Opt Header; SEQ_Cnt = 0000; | | L... |
| FCH 0005 | 05 A6 00 5F | OX_Id = 05A6; RX_Id = 005F; | | .¦._ |
| FCH 0006 | 00 00 00 00 | PARA = 00000000; | | .... |
| Pld 0001 | CA CA CA CA | | | ÊÊÊÊ |
| Pld 0002 | CA CA CA CA | | | ÊÊÊÊ |
| Pld 0003 | 00 53 00 00 | | | .S.. |
| Pld 0004 | 00 48 00 00 | | | .H.. |
| Pld 0005 | 00 68 00 00 | | | .h.. |
| Pld 0006 | 00 00 90 00 | | | ..□. |

## 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 77** Setting Filters to Isolate IDs



| PARA | Bytes | D_Id | S_Id | OX_Id | SEQ_Id | SEQ_Cnt |
|---|---|---|---|---|---|---|
| 0000B000 | 2084 | 610700 | 610300 | 05B8 | B9 | 0016 |
| 0000B800 | 2084 | 610700 | 610300 | 05B8 | B9 | 0017 |
| 0000C000 | 2084 | 610700 | 610300 | 05B8 | B9 | 0018 |
| 0000C800 | 2084 | 610700 | 610300 | 05B8 | B9 | 0019 |
| 0000D000 | 2084 | 610700 | 610300 | 05B8 | B9 | 001A |
| 0000D800 | 2084 | 610700 | 610300 | 05B8 | B9 | 001B |
| 00000000 | 48 | 610700 | 610300 | 05A6 | BF | 0000 |
| 0000E000 | 2084 | 610700 | 610300 | 05B8 | B9 | 001C |
| 0000E800 | 2084 | 610700 | 610300 | 05B8 | B9 | 001D |
| 0000F000 | 2084 | 610700 | 610300 | 05B8 | B9 | 001E |
| 0000F800 | 2084 | 610700 | 610300 | 05B8 | B9 | 001F |
| 00000000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0000 |
| 00000800 | 2084 | 610300 | 610700 | 05A6 | 4C | 0001 |
| 00000000 | 48 | 610700 | 610300 | 05B8 | B8 | 0000 |
| 00001000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0002 |

```
FC  0001    BC XX XX XX
FCH 0001    XX 61 03 00
FCH 0002    XX XX XX XX
FCH 0003    XX XX XX XX
FCH 0004    XX XX XX XX
FCH 0005    XX XX XX XX
FCH 0006    XX XX XX XX
Pld 0001    XX XX XX XX
Pld 0002    XX XX XX XX
Pld 0003    XX XX XX XX
Pld 0004    XX XX XX XX
Pld 0005    XX XX XX XX
Pld 0006    XX XX XX XX
Pld 0007    XX XX XX XX
Pld 0008    XX XX XX XX
Pld 0009    XX XX XX XX
```

```
Fibre Channel Frame
  SOF
  Fibre Channel Header
    RCtl
    D_Id = 610300
    CS_CTL
    S_Id
    Type
    F_Ctl
    SEQ_Id
    DF_Ctl
    SEQ_Cnt
    OX_Id
    RX_Id
    PARA
```

```
BC XX XX XX     Any EOF     Any End of Frame
```

```
FCH 0001    01 61 03 00    RCtl = FC4SData; D_Id = 610300;
FCH 0002    00 61 07 00    CS_CTL = 00; S_Id = 610700;
FCH 0003    08 01 00 08    Type = SCSI FCP; RX/OX = OX; S_C = S_C_I;
FCH 0004    4C 00 00 00    SEQ_Id = 4C; DF_Ctl = No Opt Header; SEQ_C
FCH 0005    05 A6 00 5F    OX_Id = 05A6; RX_Id = 005F;
FCH 0006    00 00 00 00    PARA = 00000000;
Pld 0001    CA CA CA CA
Pld 0002    CA CA CA CA
Pld 0003    00 53 00 00
Pld 0004    00 48 00 00
Pld 0005    00 68 00 00
Pld 0006    00 00 90 00
```

Show all Events like this
🟩 D_Id 610300
🟩 S_Id 610300

Except for Events like this

Ready | Ordered sets ✗ | Filters ✓ | Quick hide ✗ | Overlapping data ✗ | Speed : 2 Gbps

# 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. 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 as shown in the following sample error log.

## 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!
```

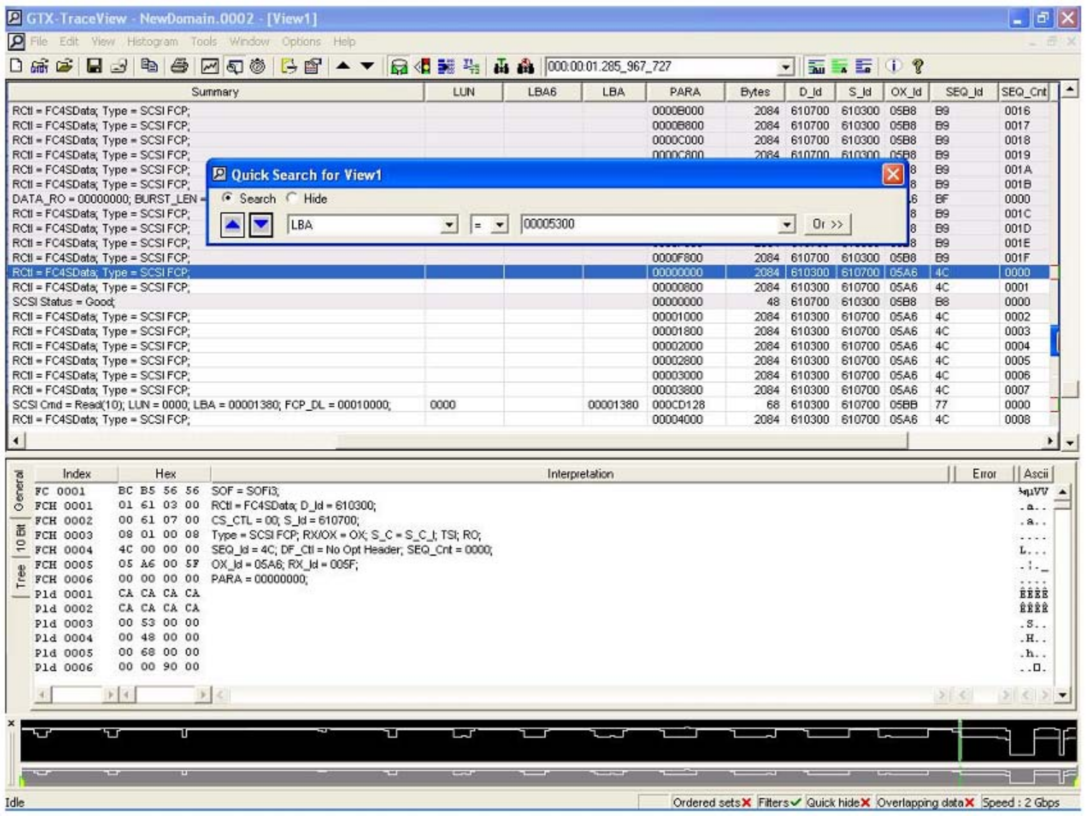
Because this example is using a physical drive as the target, the corrupt LBA reported in the error log is accurate and can be used to locate the write and read commands in the trace. Note that the corrupt LBA refers to the start of the I/O request and not necessarily the start of the data corruption 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. See Figure 78.

**Figure 78** Backward Search for Last Command to the LBA



| Summary | LUN | LBA6 | LBA | PARA | Bytes | D_Id | S_Id | OX_Id | SEQ_Id | SEQ_Cnt |
|---|---|---|---|---|---|---|---|---|---|---|
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000B000 | 2084 | 610700 | 610300 | 05B8 | B9 | 0016 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000B800 | 2084 | 610700 | 610300 | 05B8 | B9 | 0017 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000C000 | 2084 | 610700 | 610300 | 05B8 | B9 | 0018 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000C800 | 2084 | 610700 | 610300 | 05B8 | B9 | 0019 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | | | | | 8 | B9 | 001A |
| RCtl = FC4SData; Type = SCSI FCP; | | | | | | | | 8 | B9 | 001B |
| DATA_RO = 00000000; BURST_LEN = | | | | | | | | 6 | BF | 0000 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | | | | | 8 | B9 | 001C |
| RCtl = FC4SData; Type = SCSI FCP; | | | | | | | | 8 | B9 | 001D |
| RCtl = FC4SData; Type = SCSI FCP; | | | | | | | | 8 | B9 | 001E |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000F800 | 2084 | 610700 | 610300 | 05B8 | B9 | 001F |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00000000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0000 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00000800 | 2084 | 610300 | 610700 | 05A6 | 4C | 0001 |
| SCSI Status = Good; | | | | 00000000 | 48 | 610700 | 610300 | 05B8 | B8 | 0000 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00001000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0002 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00001800 | 2084 | 610300 | 610700 | 05A6 | 4C | 0003 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00002000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0004 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00002800 | 2084 | 610300 | 610700 | 05A6 | 4C | 0005 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00003000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0006 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00003800 | 2084 | 610300 | 610700 | 05A6 | 4C | 0007 |
| SCSI Cmd = Read(10); LUN = 0000; LBA = 00001380; FCP_DL = 00010000; | 0000 | | 00001380 | 000CD128 | 68 | 610300 | 610700 | 05BB | 77 | 0000 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00004000 | 2084 | 610300 | 610700 | 05A6 | 4C | 0008 |

| Index | Hex | Interpretation | Error | Ascii |
|---|---|---|---|---|
| FC 0001 | BC B5 56 56 | SOF = SOFi3; | | ¾µVV |
| FCH 0001 | 01 61 03 00 | RCtl = FC4SData; D_Id = 610300; | | .a.. |
| FCH 0002 | 00 61 07 00 | CS_CTL = 00; S_Id = 610700; | | .a.. |
| FCH 0003 | 08 01 00 08 | Type = SCSI FCP; RX/OX = OX; S_C = S_C_I; TSI; RO; | | .... |
| FCH 0004 | 4C 00 00 00 | SEQ_Id = 4C; DF_Ctl = No Opt Header; SEQ_Cnt = 0000; | | L... |
| FCH 0005 | 05 A6 00 5F | OX_Id = 05A6; RX_Id = 005F; | | .¦._ |
| FCH 0006 | 00 00 00 00 | PARA = 00000000; | | .... |
| Pld 0001 | CA CA CA CA | | | ÊÊÊÊ |
| Pld 0002 | CA CA CA CA | | | ÊÊÊÊ |
| Pld 0003 | 00 53 00 00 | | | .S.. |
| Pld 0004 | 00 48 00 00 | | | .H.. |
| Pld 0005 | 00 68 00 00 | | | .h.. |
| Pld 0006 | 00 00 90 00 | | | ..□. |

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
```

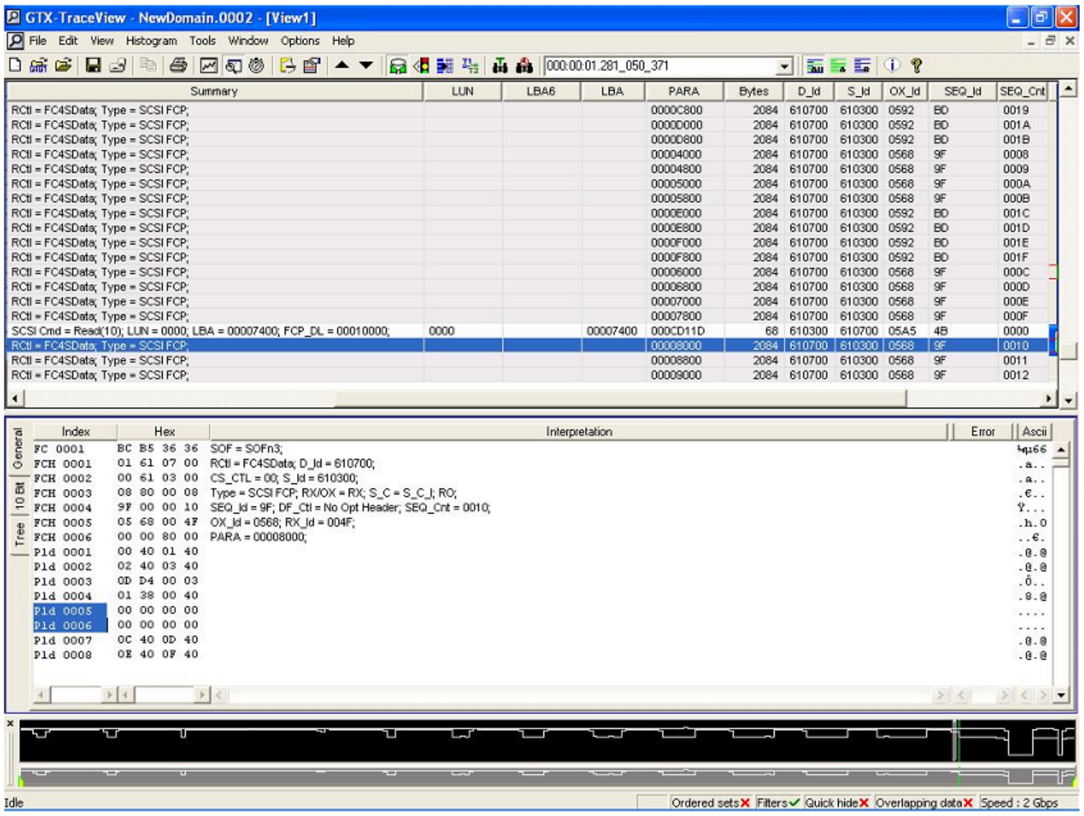
# Finding the Corrupt Data Frame

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 79.

**Figure 79** Data Frame in TraceView



GTX-TraceView - NewDomain.0002 - [View1]

File Edit View Histogram Tools Window Options Help

000:00:01.281_050_371

| Summary | LUN | LBA6 | LBA | PARA | Bytes | D_Id | S_Id | OX_Id | SEQ_Id | SEQ_Cnt |
|---|---|---|---|---|---|---|---|---|---|---|
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000C800 | 2084 | 610700 | 610300 | 0592 | BD | 0019 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000D000 | 2084 | 610700 | 610300 | 0592 | BD | 001A |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000D800 | 2084 | 610700 | 610300 | 0592 | BD | 001B |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00004000 | 2084 | 610700 | 610300 | 0568 | 9F | 0008 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00004800 | 2084 | 610700 | 610300 | 0568 | 9F | 0009 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00005000 | 2084 | 610700 | 610300 | 0568 | 9F | 000A |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00005800 | 2084 | 610700 | 610300 | 0568 | 9F | 000B |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000E000 | 2084 | 610700 | 610300 | 0592 | BD | 001C |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000E800 | 2084 | 610700 | 610300 | 0592 | BD | 001D |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000F000 | 2084 | 610700 | 610300 | 0592 | BD | 001E |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 0000F800 | 2084 | 610700 | 610300 | 0592 | BD | 001F |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00006000 | 2084 | 610700 | 610300 | 0568 | 9F | 000C |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00006800 | 2084 | 610700 | 610300 | 0568 | 9F | 000D |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00007000 | 2084 | 610700 | 610300 | 0568 | 9F | 000E |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00007800 | 2084 | 610700 | 610300 | 0568 | 9F | 000F |
| SCSI Cmd = Read(10); LUN = 0000; LBA = 00007400; FCP_DL = 00010000; | 0000 | | 00007400 | 000CD11D | 68 | 610300 | 610700 | 05A5 | 4B | 0000 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00008000 | 2084 | 610700 | 610300 | 0568 | 9F | 0010 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00008800 | 2084 | 610700 | 610300 | 0568 | 9F | 0011 |
| RCtl = FC4SData; Type = SCSI FCP; | | | | 00009000 | 2084 | 610700 | 610300 | 0568 | 9F | 0012 |

```
Index      Hex           Interpretation                                                 Error  Ascii
FC  0001   BC B5 36 36   SOF = SOFn3;                                                          ¼µ66
FCH 0001   01 61 07 00   RCtl = FC4SData; D_Id = 610700;                                       .a..
FCH 0002   00 61 03 00   CS_CTL = 00; S_Id = 610300;                                           .a..
FCH 0003   08 80 00 08   Type = SCSI FCP; RX/OX = RX; S_C = S_C_I; RO;                         .€..
FCH 0004   9F 00 00 10   SEQ_Id = 9F; DF_Ctl = No Opt Header; SEQ_Cnt = 0010;                  Ÿ...
FCH 0005   05 68 00 4F   OX_Id = 0568; RX_Id = 004F;                                           .h.O
FCH 0006   00 00 80 00   PARA = 00008000;                                                      ..€.
Pld 0001   00 40 01 40                                                                         .@.@
Pld 0002   02 40 03 40                                                                         .@.@
Pld 0003   0D D4 00 03                                                                         .Ô..
Pld 0004   01 38 00 40                                                                         .8.@
Pld 0005   00 00 00 00                                                                         ....
Pld 0006   00 00 00 00                                                                         ....
Pld 0007   0C 40 0D 40                                                                         .@.@
Pld 0008   0E 40 0F 40                                                                         .@.@
```

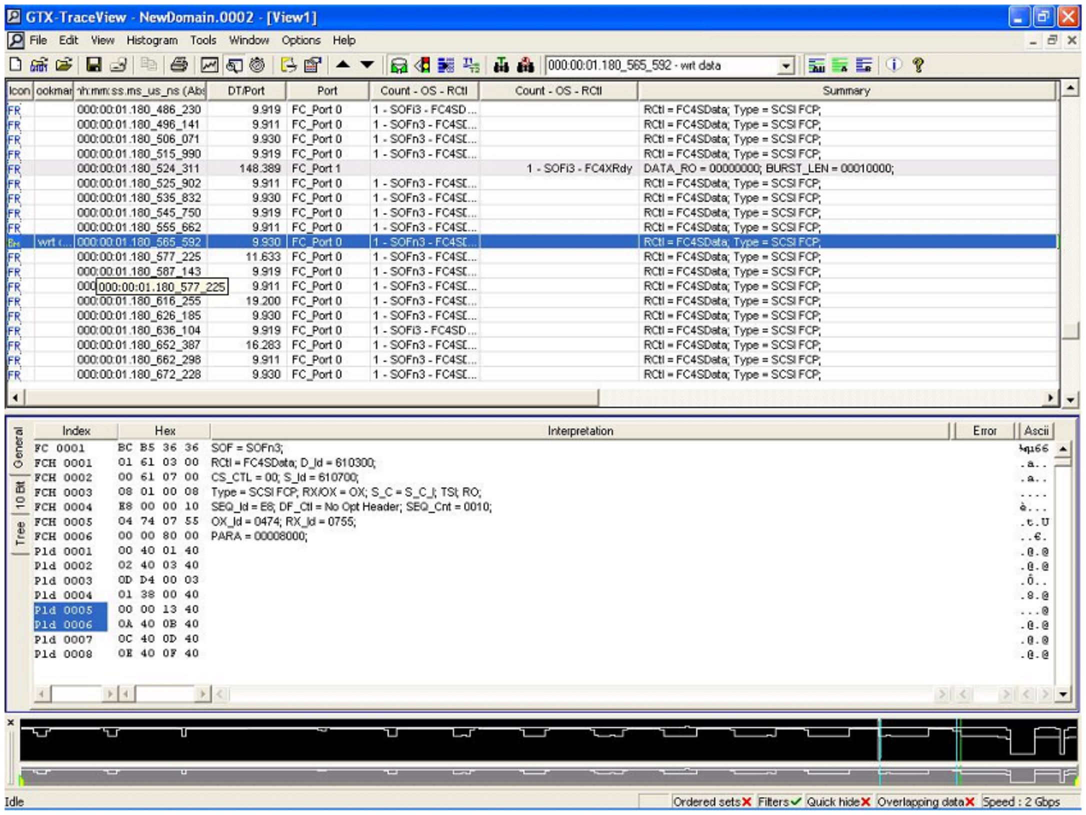
In this data frame, we find the corrupted data listed in the error log – two words of zeroes instead of the expected data.

```
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. See Figure 80.

**Figure 80** Verifying the Expected Data


GTX-TraceView - NewDomain.0002 - [View1]
File Edit View Histogram Tools Window Options Help
000:00:01.180_565_592 - wrt data
General
10 Bit
Tree

| Icon | ookmar | hh:mm:ss.ms_us_ns (Abs | DT/Port | Port | Count - OS - RCtl | Count - OS - RCtl | Summary |
|---|---|---|---|---|---|---|---|
| FR | | 000:00:01.180_486_230 | 9.919 | FC_Port 0 | 1 - SOFi3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_496_141 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_506_071 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_515_990 | 9.919 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_524_311 | 148.389 | FC_Port 1 | | 1 - SOFi3 - FC4XRdy | DATA_RO = 00000000; BURST_LEN = 00010000; |
| FR | | 000:00:01.180_525_902 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_535_832 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_545_750 | 9.919 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_555_662 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| Bm | wrt (... | 000:00:01.180_565_592 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_577_225 | 11.633 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_587_143 | 9.919 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000 000:00:01.180_577_225 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_616_255 | 19.200 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_626_185 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_636_104 | 9.919 | FC_Port 0 | 1 - SOFi3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_652_387 | 16.283 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_662_298 | 9.911 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |
| FR | | 000:00:01.180_672_228 | 9.930 | FC_Port 0 | 1 - SOFn3 - FC4SD... | | RCtl = FC4SData; Type = SCSI FCP; |

| Index | Hex | Interpretation | Error | Ascii |
|---|---|---|---|---|
| FC 0001 | BC B5 36 36 | SOF = SOFn3; | | ¼µ66 |
| FCH 0001 | 01 61 03 00 | RCtl = FC4SData; D_Id = 610300; | | .a.. |
| FCH 0002 | 00 61 07 00 | CS_CTL = 00; S_Id = 610700; | | .a.. |
| FCH 0003 | 08 01 00 08 | Type = SCSI FCP; RX/OX = OX; S_C = S_C_I; TSI; RO; | | .... |
| FCH 0004 | E8 00 00 10 | SEQ_Id = E8; DF_Ctl = No Opt Header; SEQ_Cnt = 0010; | | è... |
| FCH 0005 | 04 74 07 55 | OX_Id = 0474; RX_Id = 0755; | | .t.U |
| FCH 0006 | 00 00 80 00 | PARA = 00008000; | | ..€. |
| Pld 0001 | 00 40 01 40 | | | .@.@ |
| Pld 0002 | 02 40 03 40 | | | .@.@ |
| Pld 0003 | 0D D4 00 03 | | | .Ô.. |
| Pld 0004 | 01 38 00 40 | | | .8.@ |
| Pld 0005 | 00 00 13 40 | | | ...@ |
| Pld 0006 | 0A 40 0B 40 | | | .@.@ |
| Pld 0007 | 0C 40 0D 40 | | | .@.@ |
| Pld 0008 | 0E 40 0F 40 | | | .@.@ |

Idle | Ordered sets ✗ | Filters ✓ | Quick hide ✗ | Overlapping data ✗ | Speed : 2 Gbps

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 transient (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 embedded 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 K "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 as shown in the "Locating the Offset" example. 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.

## 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...
```

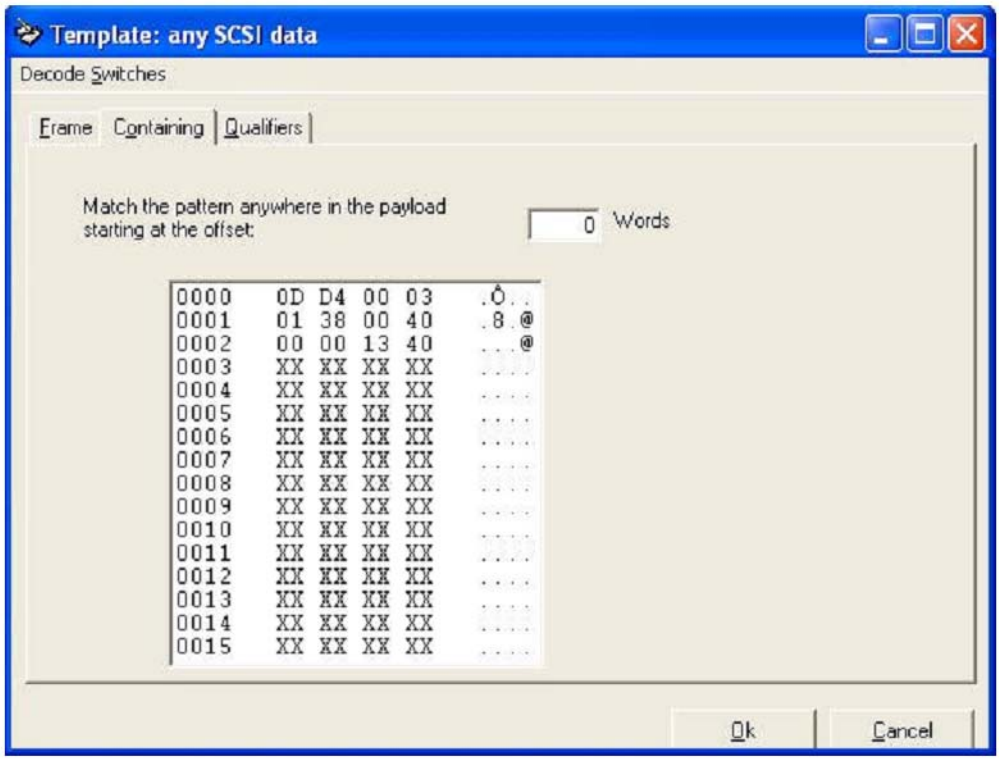
## Using I/O Signatures

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. See Figure 81.

**Figure 81** Using I/O Signature for a Search Filter in TraceView



Match the pattern anywhere in the payload starting at the offset: 0 Words

```
0000    0D D4 00 03    .Ô..
0001    01 38 00 40    .8.@
0002    00 00 13 40    ...@
0003    XX XX XX XX    ....
0004    XX XX XX XX    ....
0005    XX XX XX XX    ....
0006    XX XX XX XX    ....
0007    XX XX XX XX    ....
0008    XX XX XX XX    ....
0009    XX XX XX XX    ....
0010    XX XX XX XX    ....
0011    XX XX XX XX    ....
0012    XX XX XX XX    ....
0013    XX XX XX XX    ....
0014    XX XX XX XX    ....
0015    XX XX XX XX    ....
```

## 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
```

D

# I/O Signatures

This appendix describes MLTT's format and use of I/O signatures. By default 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. There are two purposes for this signature.

The first purpose is it is a vital component of data integrity checking in MLTT. A constantly changing data stream is needed for catching cases of out-of-order write completions or stale data corruptions. Stale data corruptions occur when 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 (without a unique I/O signature) was used for every write operation, it would be impossible to detect if the read data returned is from the immediately preceding write.

The second reason is due to I/O signatures being utilized in debugging issues discovered with MLTT. The information embedded in the signature correlates with log files from a test run to determine the initiator, target, I/O position within a file, LBA, and other information needed for debugging.

MLTT embeds a signature at the beginning of every logical block. The size is device dependent, so if the device's logical block size is 512 bytes, the MLTT will embed a signature every 512 bytes. 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.)

## Offsets

Below provides more information about what is stored at each offset.

**Offset 0:** Payload data (i.e. data pattern) (8 bytes)

This 8 bytes of data pattern content

**Offset 8:** Session ID (2 bytes)

By default, the 2 byte session ID is the last 2 ASCII characters of the target name.

· E.g. Windows “[file://./PhysicalDrive1]\\.\PhysicalDrive1” -> session ID “e1”, ASCII hex code 65 31

· E.g. Linux “/dev/sdb” -> session ID “db”, ASCII hex code 64 62

In some instances of running to multiple disks, data corruption can occur in the form of data being returned from the wrong drive. To verify this, check the data signatures in MLTT to see what the intended target is.

**Offset 10:** Thread number (2 bytes)

This is the process-global I/O thread number of the thread that wrote this data. This information is useful for ensuring that the returned data is not from another thread’s I/O area.

**Offset 12:** Loop number (1 byte)

This is the sequential loop number of the current write.

This is one way to determine if the returned data is stale - i.e. data from a previous loop or a previous test run - when doing sequential access test.

**NOTE:**
Loop count is incremented at the completion of one sequential pass of the test area. There is no such concept for random access I/O and, as a result, loop count will always be 0 when doing random access I/Os this is always 0.

**NOTE:**
Loop count is 0-based meaning that the first loop is loop #0.

**Offset 13:** Block number (3 bytes)

A block is a unit of a buffer-sized area within a thread's I/O area. E.g., for "-b64k", a block is a 64KB of area within a thread's test area corresponding to 1 I/O request. The block number is an ordinal of buffer-sized blocks in sequence from the start of the thread's I/O area to the end.

- Thread_io_area_size / buffer_size + 1

For random-access I/O, the block number has no meaning.

**Offset 16:** LBA (4 bytes)

Least-significant 4 bytes of LBA corresponding to this logical block.

**Offset 20:** Optional 1-second resolution timestamp (4 bytes)

If the timestamp option is specified ("-U"), the ANSI C "current time" timestamp is embedded.

**Offset 24:** Optional milliseconds (2 bytes)

If "-Um" is specified, in addition to the 4 byte 1-second resolution timestamp, additional milliseconds are appended

Remaining bytes in this logical block are data pattern content.

## Example of I/O Signature:

Shown below is th hexdump of written data from "maim -Q2 -b1k 4K -i3 -I1 -y0x77 -x1m -f\\.\physicaldrive1"

- 4 1K "blocks" written to 4KB test area (8 logical blocks)
- Fill with "0x77"
- Starting offset 1MB
- 3 sequential loops ("FOPS" – "file operations")

```
00100000  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 01  |wwwwwwwwe1......|
00100010  00 00 08 00 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100020  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100200  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 01  |wwwwwwwwe1......|
00100210  00 00 08 01 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100220  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100400  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 02  |wwwwwwwwe1......|
00100410  00 00 08 02 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100420  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100600  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 02  |wwwwwwwwe1......|
00100610  00 00 08 03 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100620  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100800  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 03  |wwwwwwwwe1......|
00100810  00 00 08 04 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100820  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100a00  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 03  |wwwwwwwwe1......|
00100a10  00 00 08 05 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100a20  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100c00  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 04  |wwwwwwwwe1......|
00100c10  00 00 08 06 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100c20  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00100e00  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 04  |wwwwwwwwe1......|
00100e10  00 00 08 07 77 77 77 77  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100e20  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
00101000
```

- Session ID: `65 31` (“e1” from “physicaldrive1”
- Thread number: `00 01`
- Loop number: `02` (i.e. “3rd loop” due to “-i3”)
- Blocks (1KB each spanning 2 LBs)

  Block #1: `00 00 01`

  Block #2: `00 00 02`

  Block #3: `00 00 03`

  Block #4: `00 00 04`

LBA of each signature:

`00 00 08 00`

`00 00 08 01`

`00 00 08 02`

`00 00 08 03`

`00 00 08 04`

`00 00 08 05`

`00 00 08 06`

`00 00 08 07`

With “-U”, 4 byte timestamp is added after the LBA part:

```
00100000  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 01  |wwwwwwwwe1......|
00100010  00 00 08 00 5b d2 20 01  77 77 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100020  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
```

With “-Um”, 2 bytes of milliseconds are appended to timestamp:

```
00100000  77 77 77 77 77 77 77 77  65 31 00 01 02 00 00 01  |wwwwwwwwe1......|
00100010  00 00 08 00 5b d2 20 f6  03 05 77 77 77 77 77 77  |....wwwwwwwwwwww|
00100020  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  |wwwwwwwwwwwwwwww|
*
```

E

# Exit Codes

This appendix shows return codes at exit that you can use in custom batch files or shell scripts and provides the exit code descriptions listed in numerical order. The topics discussed in this appendix are as follows:

## Using 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.)

F

# Architecture Bandwidths

Listed in this appendix are all of the architecture that are supported by Medusa Labs Test Tools Suite. Bandwidth capabilities are also listed below each architecture.

## 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

**Theoretical Bandwidth, Full-Duplex (Gb/s)**

| Link Width | X1 | X2 | X4 | X8 | X16 |
|---|---|---|---|---|---|
| **Generation 1** | 0.5 | 1.0 | 2.0 | 4.0 | 8.0 |
| **Generation 2** | 1.0 | 2.0 | 4.0 | 8.0 | 16.0 |
| **Generation 3** | 2.0 | 3.9 | 7.9 | 15.8 | 31.6 |

## Fibre Channel (Full Duplex)

1Gb - 200 MB/sec

2Gb - 400 MB/sec

4Gb - 800 MB/sec

8Gb - 1600 MB/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

## NVMe

PCIe Gen 3.0 - 8.0 GT/s per lane

PCIe Gen 4.0 - 16.0 GT/s per lane

**NOTE**
NVMe bandwidths are dependent on the underlying transport - i.e. PCIe.

G

# Sock Test Tool

This appendix provides information about the Sock Test Tool. The topics discussed in this appendix are as follows:

## Sock Notes

"Sock" for TCP and UDP I/O generation shares the same I/O engine and application logic with "pain" and "maim" because it was implemented by adding the "socket" endpoint type beneath the existing I/O engine and adding the peer-to-peer association logic ("open") above the application logic. This allowed sock to inherit almost all the existing features: e.g. data patterns and comparison, multiple threads (a thread per connection), common logging and event and error handling, etc,

On the other hand, this also means that sock has inherited the file (or disk) I/O paradigm as well which can seem awkward since TCP and UDP I/O generation is done only on the wire with no persistent backing store.

- "File size". This is obviously virtual and imaginary. This sets an imaginary I/O area of given size per thread. The imaginary lay out of this I/O area is as described for pain/maim.
  - "Sequential FOP". Even though "file size" I/O area is virtual, "FOP" has the same meaning as with pain/maim.
    - Serves to determine the "-i" limit for exit condition
    - Serves to determine the "forward" pattern and "reverse" pattern switch point
  - "Sequential-access". Without actual backing store, this is a virtual concept within the file size I/O area.
  - "Random-access". Without actual backing store, this is a virtual concept within the file size I/O area.
- "Buffer size, -b". Think of this as the application-level message boundary.
  - A application-level message of "buffer size" length will be packetized by the TCP I/O stack in a manner that is invisible to "sock".
  - "IOPS" (I/Os per second). Each transfer of "buffer size" length of data is counted as "1 I/O". There's no deterministic way to associate a sock "IOPS" with, say, "TCP packets per second" (although, if you keep the buffer size sufficiently small, 1 buffer size transfer can correspond to 1 TCP or UDP packet).
  - For UDP, the datagram format limits the maximum datagram size due to the 16-bit size field (header + payload = 65535 bytes or less). Sock imposes a maximum buffer size of 63.5KB for UDP traffic.
  - "Logical Block size". Sock uses the arbitrary 512 bytes as the imaginary logical block size, and, as with pain/maim, all sizes, including "-b" buffer size, must be an exact multiple of this logical block size. However, "sock transaction" mode allows a more arbitrary buffer size (see "Sock Transaction Mode" on page 396).

The default read-write mode for sock is always immediate "echo" (or "read-back") of buffer-sized amount of data. E.g.

```
sock -b32k
```

1 Sock: send 32KB of data to peer

2 Peer: wait until it receives all 32KB of data from sock; upon completion, echo (send back) the received 32KB of data to sock.

3 Sock: wait until it receives all 32KB of data from peer; upon completion, perform data comparison

The application-level message boundary is enforced by both sock and the peer. The buffer size (“`-b32k`” in this example) is agreed upon by both sides during the initialization process. I.e. the peer will not initiate the echo until all 32KB of data is received from sock.

When using in read-only or write-only mode, where the traffic per connection is only in 1 direction, there’s really no application-level message boundary being enforced because the end effect is that one side is simply sending as quickly as it can while the other side is consuming as quickly as it can.

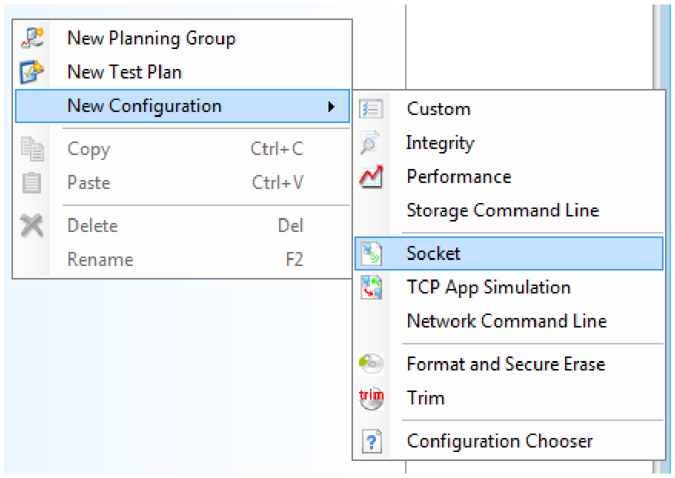
**NOTE**
In sock, there’s no “initial write-once pass” variety of read-only mode.

When using the I/O profile mode to specify a randomized mix of read and writes (“`-%r/w`”), sock enforces the per-read and per-write buffer-sized message boundary. However, the remote peer side is constantly putting data on the wire and simultaneously consuming (receiving) data from the wire. This is due to the implementation detail that only sock is aware of the randomized read/write sequence while the remote peer is not. Therefore, the remote peer assumes that there may be data from sock to consume any given moment, and it also assumes sock may be blocked on data to consume any given moment. The peer is in constantly simultaneous send loop and receive loop in order to avoid a deadlock.

In CLI, sock is invoked as follows:

- TCP: “`sock`” or “`sock -m30`”
- UDP: “`sock -m31`”
- ASIDE: “`pain -m30`” and “`pain -m31`” also works (synchronous I/O – “`pain`” – using socket endpoints).

In the GUI, sock is invoked as a “Socket” configuration type:



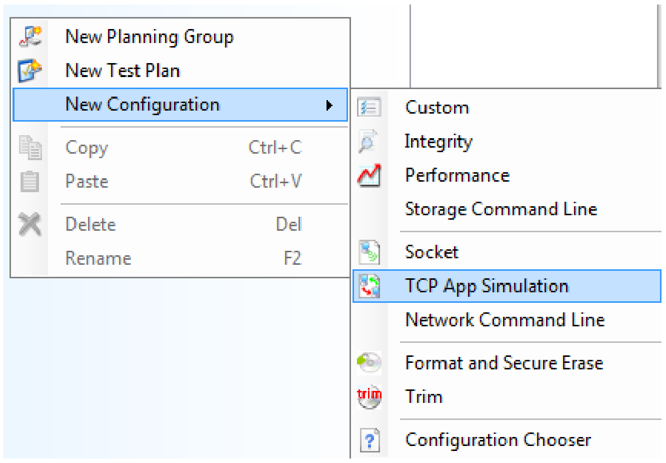
## Sock Transaction Mode

Sock has a “transaction mode” which is an I/O profile mode that is unique to it – and probably one of the least used and understood features. It is designed to simulate a typical application-level transaction loads. A prominent application-level transaction example is the HTTP “GET” request sent from a Web browser to a Web server, and the Web server sending back the response with data requested in the URL (e.g. an HTML document). A SQL query sent from a database client and the data returned by the SQL server is another example. This 1 pair of “request-response” is 1 application-level transaction. In transaction mode, sock acts as the server while remote peers act as clients.

In the CLI, the sock transaction mode is invoked with “`sock -%T`”, and the per-transaction payload is specified using the “-%r” (request) and “-%w” (response) specifiers.

In the GUI, you create a “TCP App Simulation” configuration.

**Figure 82** TCP App Simulation Menu



Creating transaction profile involves specifying at least 1 request size, and at least 1 response size, and, optionally, number of transactions per connection.

- Enable transaction mode with "`-%T`".

  Optionally, specify the session length. "`-%T4`" says "4 transactions per connection" (i.e. every 4 transactions, the connection will be closed and a new connection will be opened for next 4 transactions). A random range can be specified. E.g "`-%T4-6`" – a random session length will be chosen from values between 4 and 6 (inclusive). If no value is specified here, same connection per thread will be used for all transactions until the end of the test.

**NOTE**

Typically, the client initiates the TCP connection to the server in real world applications. However, sock's inherent design is for sock to initiate the connection to the peer, and this transaction mode was built on top of that design. Therefore, in sock transaction mode, you end up with the "server" initiating connection to the "client".

- Request size specification: "`-%r`" (sock, the server, receives requests from remote peers – the clients).

  A simple example is "`-%r100@100b`", which specifies "all requests are 100 bytes in length".

  Another example is "`-%r50@100b,r50@1k-2k`", which states "50% of the requests are 100 bytes in length, the other 50% of the requests can be between 1k and 2k in length".

  Currently, the sock transaction mode is the only I/O mode in MLTT that allows an arbitrary buffer size that is not a multiple of "logical block size". The transaction mode uses the first 8 bytes of payload data for house-keeping message between sock and peers; therefore, the minimum buffer size you can use for request or response is 8 bytes.
- Response size specification: "`-%w`" (sent from sock to remote peers)

Similarly, you specify the probability weight and the size. E.g. "`-%w100@32k-64k`" (all responses are between 32K and 64k in length).

Recall that for normal "`-%r`" and "`-%w`" profile specification, the sum of all "`-%r`" and "`-%w`" probability weights are used to determine the probability of each spec. However, in transaction mode, the sum of only the "`-%r`" weights are used to calculate the request probabilities, while the sum of only the "`-%w`" weights are used to calculate the response probabilities.

Looking at an example of loading a front page of "http://www.bogus-acme.net/" using a Web browser. It might follow a work flow like this:

- Browser: request via HTTP GET "index.html"
- Server: send "index.html"
- Browser: parses in stream the HTML document which contains about 16 embedded images

- Browser: opens 3 more connections (so 4 connections total with the original connection). With HTTP keep-alive and pipe-lining per connection to reduce connection latency, the browser fetches 4 images per connection (each requiring a pair of HTTP GET and response transactions). The images are 16KB to 64KBs in length. Each GET request is about 100 to 200 bytes in lengths.

If the HTTP GET-response statistics were gathered using the browser debug mode or the HTTP server logs. The simplest transaction mode simulation profile might look like this to simulate a repeated fetch of the front page and the embedded images:

```
sock -t4 -%T4 -%r100@100b-200b -%w100@16k-64k
```

The 4 threads (“`-t4`”) creates 4 concurrent connections (transaction pipelines), “`-%T4`” reflects about 4 resources that were fetched per connection during the course of loading the page and embedded resources (images). Each thread will do the following:

- Open connection.
- Wait for a request from peer (client), 100 bytes to 200 bytes in length.
- When all of the request is received, randomly pick a buffer size between 16K and 64K and send the response -> 1 transaction done.
- Repeat for 3 more transactions.
- Close connection
- Open connection
- Do next 4 transactions.
- Close connection.
- etc …

Or, let’s say you break down the transaction details more.

- First GET message was 100 bytes.
- “index.html” document returned was 8KB.
- The 16 GET messages to fetch the images were 150-200 bytes each.
- 4 images were 16KBs each.
- 4 images were 32KBs each.
- 8 images were 64KBs each.

With that, you fine tune the profile as:

```
sock -t4 -%T4 -%r1@100b,r16@150b-200b -%w1@8k,w4@16k,w4@32k,w8@64k
```

I.e.

- “`-%r1@100b`” – first 100 byte GET request
- “`-%r16@150b-200b`”– the 16 GET requests per image, 150 bytes to 200 bytes each
- “`-%w1@8k`” – the 8KB “index.html”
- “`-%w4@16k`” – the 4 images, 16KB each

- "`-%w4@32k`" – the 4 images, 32KB each
- "`-%w8@64K`" – the 8 images, 64KB each

The corresponding work flow per thread (connection):

- Sock: open connection
- Peer: randomly picks a request size, 100B or a random value between 150B to 200B, according to the probability specified with "-%r". Send request.
- Sock: receive request. Randomly pick a response size, 8k, 16k, 32k, or 64k, according to the probability specified with "-%w". Send response -> 1 transaction done.
- etc …

## Additional Information

- This is not an emulation. Sock does not actually implement a given protocol (e.g. HTTP). It also does not try to emulate whatever initialization or hand-shake sequence a particular application-level protocol might define.
- It is a simulation of load. Profiles can be devised using statistical analysis of actual application-level transaction logs.

  If you want to synthesize a profile using packet-level capture, be sure to aggregate all packets belonging to each application-level request, and do the same for each application-level response, rather than looking at packets independently.
- Note how requests and responses are picked randomly – the sock transaction mode does not try to match a specific request to a specific response as a pair.
- Instead of IOPS, "`sock -%T`" shows TPS (transactions per second) for request-response pairs.

# H

# Journal Verification

This section discusses the limitations of MLTT's journal verification mode implementation. Due to these limitations, when using the journal verification feature for the intended use-case of testing sudden power-loss scenarios, the state of the write operations recorded in the journal could be ambiguous. A recorded state in the journal of write operations that was dispatched to the device near the moment of power-loss does not necessarily show if the write operation succeeded or failed. This section explores in detail the nuances of these ambiguities and other pitfalls to consider.

## Limitations

When MLTT runs I/O to a target with journal recording enabled, each write operation to the target goes through the following steps:

1. Write in the journal file the details of the write operation that is about to be submitted to the target. A record for a single write operation contains:

    a. I/O offset (LBA)

    b. I/O size ("transfer size")

    c. Microsecond timestamp of I/O submission

    d. Write state "QUEUED"

2. Submit the I/O request to the OS

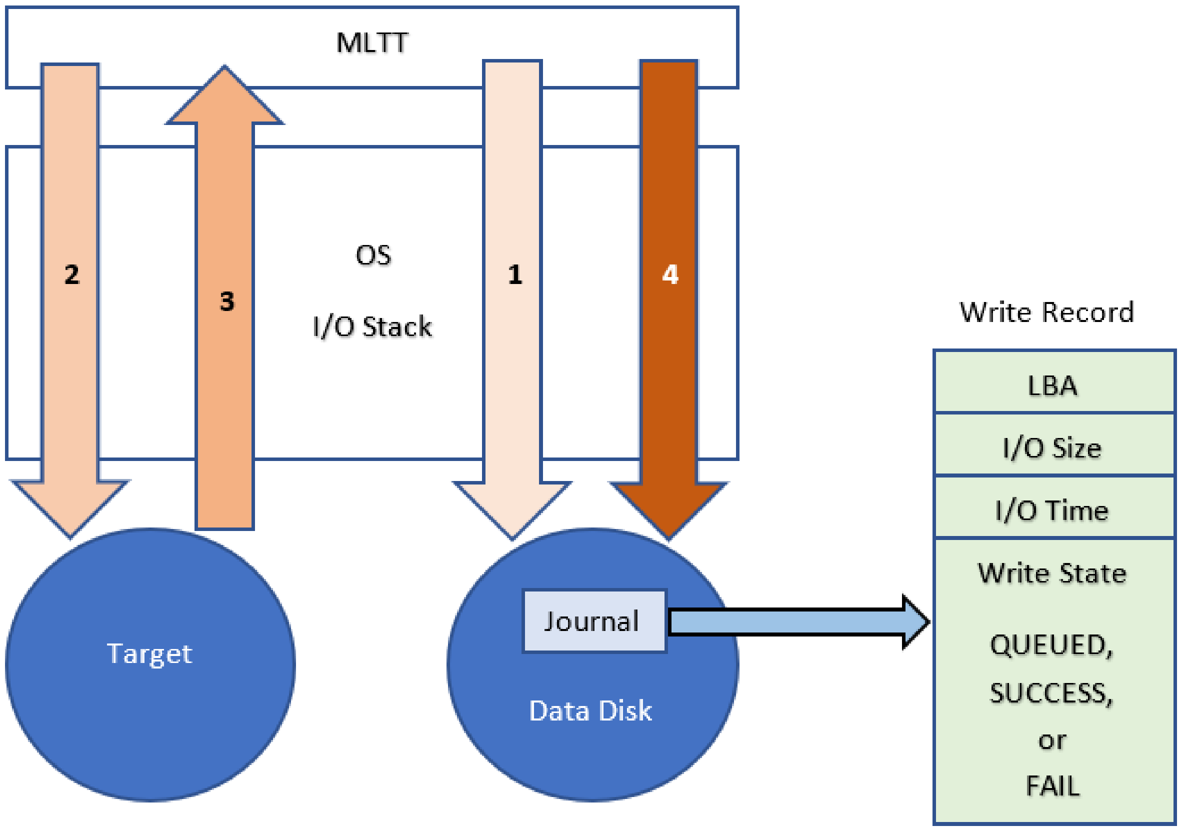
3. Wait for the OS to return an I/O completion status for the submitted I/O request

4. Write the updated state in the corresponding record in the journal file - i.e. the record created and set to "QUEUED" state and written to the journal file in step #1.

a. Write state “SUCCESS” if the I/O completion status returned by the OS indicates that the write was successful

b. Write state “FAIL” if the I/O completion status returned by the OS indicates that the write had failed

The diagrams shown below illustrate the four steps within the context of the separate layers of the DUT host: MLTT (application layer), the OS, I/O stack, and the I/O devices (the test target and the data disk).



When power is suddenly removed from the DUT during any of the four steps, the state of the write operation, as recorded in the journal, is no longer guaranteed.

This is due to the fact that the journal record updates are disk write operations themselves. As shown in the four steps above, keeping track of a write operation to the target requires two writes to the journal file. One write is used before the write operation to the target to put the record in “QUEUED” state, and another write to the journal file, after the target I/O completion, that sets the completion status to “SUCCESS” or “FAIL”. The write operation to the target and the two corresponding writes to the journal file are not executed as one atomic operation. The three writes executed as three separate, serial write requests made to the OS.

When the power-loss interrupt occurs, the write status recorded in the journal does not indicate the precise moment at which the interrupt occurred. Furthermore, the write operations to the journal are just as susceptible to being interrupted as the write operations to the target device.

## Write State Ambiguities

Described below are some possible implications of a recorded write state when MLTT examines it after rebooting the DUT host following the power-loss interrupt.

If the recorded write state for an attempted write operation to the target is in a “SUCCESS” state, then MLTT is certain that the previous process had completed all four steps of a journaled write operation to the target and the journal file. However, if the record is in a “QUEUED” state, it does not necessarily mean that the write operation to the target had not occurred. Furthermore, if the record is in a “FAIL” state, it does not necessarily mean that the write operation to the target had failed. Therefore, any recorded write state that is not “SUCCESS” is ambiguous. In other words, a “QUEUED” or a “FAIL” state does not indicate whether or not the write operation had actually occurred on the target device.

### QUEUED State Ambiguities

If the record for an attempted write to the target device is in a “QUEUED” state, the only certainty is that MLTT had successfully updated the journal file and marked the record as a “QUEUED” state in step #1. However, any one of the following could have happened after step #1 depending on when the power-loss interrupt occurred:

A. Power-loss interrupt occurred right after step #1, before MLTT could transition to step 2 to submit the write command and payload data to the OS

B. MLTT transitioned to step #2 and successfully submitted the write request to the OS; however, power-loss interrupt occurred before the OS submitted the write command to the device

C. The OS submitted the write command to the device; however, power-loss interrupt occurred before the device could successfully transfer the payload data to the medium

D. The device successfully processed the write command and transferred the payload data to the medium - i.e. a successful write had occurred; however, power-loss interrupt occurred before the device could return the “SUCCESS” status to the OS

E. The device encountered an error – e.g. a media error; however, power-loss interrupt occurred before the device could return the “FAIL” status to the OS

F. The device passed the “SUCCESS” or “FAIL” status to the OS; however, power-loss interrupt occurred before the OS could transition to step #3 to return the status to MLTT

G. MLTT got the “SUCCESS” or “FAIL” status from the OS in step 3; however, power-loss interrupt occurred before MLTT could transition to step 4 to update the record in the journal file with the “SUCCESS” or “FAIL” status.

The attempted write command whose record in the journal file is in a “QUEUED” state could have never reached the device, or could have transferred successfully - i.e. would have been in “SUCCESS” state if there was no power-loss interrupt, or could have failed processing at the device - i.e. would have been in a “FAIL” state if there was no power-loss interrupt.

### FAIL State Ambiguities

Under normal operating circumstances, there should be no ambiguity when a write attempt results in a “FAIL” status. However, if the write attempt was interrupted via power-loss, then the “FAIL” status does not necessarily indicate that the write command processing on the device had failed to transfer the data to the medium. As previously stated in the “QUEUED” state ambiguities explanation, the power-loss interrupt can occur after the successful transfer of data to the medium but before the device can return the “SUCCESS” status to the OS.

Any of the following scenarios listed below could happen in the event of a “FAIL” state depending on the timing of the power-less interrupt:

A. Power-loss interrupt occurs just as the device is about to return the “SUCCESS” or “FAIL” status to OS.

B. OS detects that the device has gone off line and passes a “FAIL” status back to MLTT as described in step #3

C. MLTT updates the record for the attempted write with “FAIL” status in the journal file.

In the event of a power-loss interruption, the “FAIL” status in the journal file is a result of the OS indicating that the device lost connection even though the write command may have actually succeeded.

## Ambiguity Resolution

The ambiguities described above have an impact on how to interpret the data comparison results during the MLTT journal verification process executed after restoring power to the DUT.

## 7.2.0

The first implementation of journal verification included in MLTT 7.2.0 handled the record states as follows:

A. “SUCCESS” status - There is absolutely no ambiguity with this state. The OS had returned the “SUCCESS” status, and MLTT had recorded the status in the journal file for the write command. If the data comparison detects mis-compare for this write, MLTT raises the “Data Corruption” error.

B. “FAIL” status - MLTT 7.2.0 did not consider the ambiguities of this status. Because the OS returned the “FAIL” status for the write, MLTT 7.2.0 treated this as if no data had transferred to the medium and did not perform data verification on these writes.

C. “QUEUED” status - MLTT 7.2.0 attempted no ambiguity resolution on writes in this state. Data comparison was performed on these writes. If verification was successful, it was counted as “OK”. However, if the verification failed, instead of raising a “Data Corruption” error it was reported as “UNDEFINED”.

There are two issues with MLTT 7.2.0. First, by ignoring writes in “FAIL” state and not performing data verification on those writes, legitimate data corruptions could be missed.

The second issue is the false data corruption error that can occur due to an undetected overwrite. For example, consider the following sequence of events:

A. Write 16 logical blocks at LBA 1000. This write is recorded as “SUCCESS” in the journal

B. Perform more writes

C. Around the time of power-loss interrupt, attempts to write 16 logical blocks at LBA 1008.

The first 8 logical blocks of write a C overwrites the last 8 logical blocks of the earlier write in A. If the later write in C is recorded as “SUCCESS”, MLTT detects the overwrite. The later write in C is verified, but the earlier write in A, because it was overwritten, is not verified in order to avoid raising false data corruption.

However, if the later write in C is a “FAIL” state, or if it is in “QUEUED” state with “UNDEFINED” verification result, MLTT 7.2.0 did not detect the overwrite in A. This caused a failed verification on the overwritten data in A and incorrectly raised a “Data Corruption” error and had to be manually resolved.

## 7.3.0

MLTT 7.3.0 implements a write detection algorithm for writes in “FAIL” state.

A. For writes in a "FAIL" state, if MLTT detects that at least 1 logical block had some data transferred to the medium successfully, MLTT performs data verification on that write.

   i. If the data verification succeeds, the write is verified as "OK"

   ii. If the data verification fails, MLTT raises a data corruption error on the write

B. If MLTT does not detect any logical block with actual data transfer, the write in "FAIL" state is reported as an "UNDEFINED" verification result

Furthermore, with this implementation, MLTT detects if a write in "FAIL" state had overwritten an earlier write and avoids raising false data corruption errors from undetected overwrite.

### 7.4.0 (7.3.0+)

MLTT 7.4.0 implements the write detection algorithm for writes in a "QUEUED" state as well. This was implemented in a out as a patch to 7.3.0 for some customers. This further eliminated false data corruptions due to overwrite by "QUEUED" writes.

## Partial Commits And "--jv-compat"

The write detection algorithms implemented in 7.3.0 and 7.4.0 provide automatic ambiguity resolution and proper detection of overwritten previous writes to avoid false data corruption. The ambiguity resolution detection, however, was not able to detect partial writes that were interrupted at the time of power-loss.

A common symptom of a write at the time of power-loss interrupt is a partial commit.

Ideally, when a queued write is interrupted by power loss, the correct processing should result in either none of the logical blocks of the transfer persisted to the medium, or all logical blocks of the transfer persisted to the medium. However, "partial commits" can also occur, where some - i.e. not all - logical blocks of the transfer in the command are persisted to the medium.

With the write detection algorithm in 7.4.0, if MLTT raises a data corruption error on a write in a "QUEUED" or "FAIL" state, it is actually detecting this partial commit on a write that occurred at the moment of power loss.

From the application level, a partial commit is a legitimate data corruption because it signifies some existing data being overwritten in an unintended manner. However, at the device level, whether or not this should be considered a defect is up to the device specification. For example, if the device specification guarantees an atomic write of N logical blocks at power-loss and if the interrupted transfer size is greater than N logical blocks, then partial commit does not violate the device specification.

In MLTT 7.2.0, these partial commits were counted as verification “UNDEFINED” rather than raising a data corruption error. In MLTT 7.4.0, you can restore this data verification accounting behavior by specifying the “--jv-compat” option (“journal verification compatibility”).

## Pitfalls

When you abruptly remove power from the DUT host, the system is susceptible to potential damage:

The OS installation can become corrupt

The journal file can become corrupt

Physical damage can occur (mechanical disk crash, electrical damage to chips, etc)

While the risk is unavoidable when performing power-loss interrupt tests involving abrupt power removal from the DUT host, there are some measures to mitigate the potential journal file corruption.

Bypassing the file system and specifying a physical device as the journal “directory” can help - e.g. “`--journal=/dev/sdd`”, “`--journal=\\.\physicaldrive5`”.

The best approach is to put the journal in a storage with a separate power source, for example:

On a shared network directory. The actual file system host is a NAS or another server.

External USB drive with its own power supply. Combine this with physical drive journal directory specification.

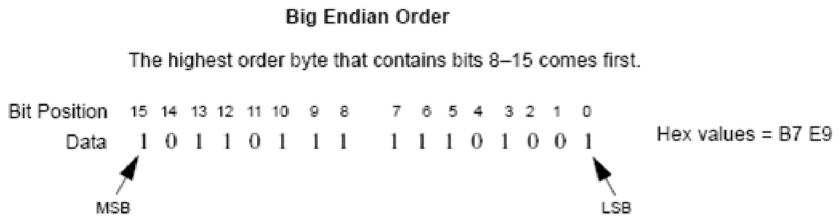
External storage with its own power supply connected via SAS or FC HBA with onboard battery. Again, combine this with physical drive journal directory specification.

In MLTT 7.4.0, when specifying the physical drive as the journal directory, during the “--journal -V” verification run, MLTT creates a copy of the journal file named “mltt.journal-device-copy” in the current directory. This can be helpful when technical support is needed because a copy of the journal can be sent for examination.

# Glossary

**B**

**Big Endian —** Big Endian is the order in which bytes are arranged. When using multiple bytes, the order must be identified. 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:


Big Endian Order
The highest order byte that contains bits 8–15 comes first.
Bit Position 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Data 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 1
Hex values = B7 E9
MSB
LSB

**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.

**D**

**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.

**F**

**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. The options -m17, -m18, --full-device all perform full device coverage.

For full device coverage, “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.

## I

**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.

## L

**LBA —** Logical block address.

## M

**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 when 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.

## Q

**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.

R

**Random Access —** Random access refers to I/O operations across a device at randomly selected offsets.

S

**Secure Erase —** The Secure Erase configuration editor erases the data on a 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

T

**T10-PI —** T10 Protection Information (PI) is a standard for protecting the integrity of data. The data is validated as it moves end-to-end through the data path from the HBA to the storage device. This is done by appending an extra 8 bytes of integrity metadata to each 512-byte sector of data. There are four device protection types or levels.

**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.

**TCP Incast —** A severe failure of TCP throughput caused by the number of servers exceeding the Ethernet switch limitations to buffer the packets being sent to a client. A microburst of TCP data is sent from several servers to a single switch resulting in the loss of data.

**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 sending 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 Trim configuration erases specified data blocks. It may be run as a target drive pre-conditioning step before running I/O tests.

W

**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

Version 7.8
July 2023
English

VIAVI Solutions

|  |  |
|---|---|
| North America: | 1.844.GO VIAVI / 1.844.468.4284 |
| Latin America | +52 55 5543 6644 |
| EMEA | +49 7121 862273 |
| APAC | +1 512 201 6534 |
| All Other Regions: | viavisolutions.com/contacts |