dt – Data Test Program

September 17th, 2025

Version 25.05

Author: Robin T. Miller

e-mail: Robin.T.Miller@gmail.com

EXTREME WARNING!!!

Use of this program is *almost* guaranteed to find
problems and cause your schedules to slip!
But seriously, dt has a proven history of finding
host, interconnect, and storage related issues, with many features
to help diagnose, triage and troubleshoot said issues. 😊

Bottom line: Adding *dt* to your toolbox will definitely
help improve the quality of your storage product (IMHO).

Contents

Overview 8
Additional Documentation 9
New Feature Summary 9
Operating Systems Supported 10
Program Options 10
Input File “if=" Option 10
Output File “of=" Option 10
SCSI Device “sdsf=” Option 11
SCSI Trigger "tdsf=" Option 11
Pattern File "pf=" Option 11
Block Size "bs=" Option 12
Log File "log[atu]=" Options 12
POSIX Asynchronous I/O "aios=" Option 13
Keepalive Alarm Time “alarm=” Option 13
Keepalive Message “*keepalive=” Options 13
Keepalive Message Format Control 14
Other Format Control Keywords 14
Buffer Alignment "align=" Option 14
File Disposition "dispose=" Option 15
Dump Data Limit "dlimit=" Option 15
Data Corruption Analysis Options 15
Dump Format "dfmt=" Option 15
Buffer Offset "boff=" Option 15
Max Bad Blocks "maxbad=" Option 16
Device Size "dsize=" Option 16
Device Type "dtype=" Option 16
Input Device Type "idtype=" Option 16
Output Device Type "odtype=" Option 16
Error Limit "errors=" Option 17
File System Test Options 17
File System Buffering Modes "bufmodes=" Option 18
Directory Path "dir=" Option 18
Directory Prefix "dirp=" Option 18
Number of Subdirectories "sdirs=" Option 18
Subdirectory Depth "depth=" Option 19
File Limit "files=" Option 19
File Frequency Flush "ffreq=" Option 20
Maximum Data "maxdata=" Option 20
Maximum Data Percentage "maxdatap=" Option 20
Maximum Files "maxfiles=" Option 20
File Plush Frequency “ffreq=” Option 21
File System Trim Frquency “fstrim_freq=” Option 21
Write Fill Pattern “fill_pattern=” Option 21
Read Prefill Pattern “prefill_pattern=”) Option 21

Terminal Flow Control "flow=" Option 21
History "history=" Option 22
History Buffer "history_bufs=" Option 22
History Block Data Size "history_bsize=" Option 22
History Data Size "hdsize=" Option 22
Record Increment "incr=" Option 22
I/O Per Second “iops=” Option 23
I/O Direction "iodir=" Option 23
I/O Mode "iomode=" Option 23
IOT Pass "iotpass=" Option 23
IOT Seed "iotseed=" Option 23
I/O Tune File "iotune=" Option 24
I/O Type "iotype=" Option 24
Minimum Record Size "min=" Option 24
Maxmimum Record Size “max=“ Option 24
Logical Block Address “lba=“ Option 25
Logical Block Size “lbs=“ Option 25
Data Limit “limit=“ Option 25
Increment Limit “Incr_limit=“ Option 25
Minimum Limit “min_limit=“ Option 26
Maximum Limit “max_limit=“ Option 26
Capacity “capacity=“ Option 26
Capacity Percentage “capacityp=” Option 26
Log File “log=“ Option 26
    Log File Format Control Strings 27
Log Prefix “logprefix=” Option 27
Error Log “error_log=” Option 27
Master Log “master_log=” Option 27
Reread File “reread_file” Option 27
Common Open Flags “flags=“ Option 27
Output Open Flags “oflags=“ Option 27
Verify Flags “vflags=“ Option 28
On Error Action “onerr=“ Option 28
No Progress Time “noprogt=“ Option 28
No Progress Time Trigger “noprogtt=“ Option 29
No Time “notime=“ Option 29
Terminal Parity Setting “parity=“ Option 29
Pass Limit “passes=“ Option 29
Pass Command “pass_cmd=“ Option 30
Data Pattern “pattern=“ Option 30
File Position “position=“ Option 30
Output File Position “oposition=“ Option 30
Prefix “prefix=“ Option 31
Multiple Processes “procs=“ Option 31
Set Queue Depth “qdepth=“ Option 31
Random I/O Offset Alignment “ralign=“ Option 32

Random I/O Data Limit “rlimit=“ Option 32
Random Seed Value “rseed=“ Option 32
Record Limit “records=“ Option 32
Read Percentage “readp=“ Option 32
Random Percentage “randp=“ Option 33
Read Random Percentage “rrandp=“ Option 33
Write Random Percentage “wrandp=“ Option 33
Run Time “runtime=“ Option 33
Retry Delay “retry_delay=“ Option 33
Script File “script=“ Option 33
Slice “slice=“ Option 34
Slices “slices=“ Option 34
Record Skip “skip=“ Option 34
Record Seek “seek=“ Option 34
Data Step “step=“ Option 35
Statistics Level “stats==“ Option 35
Stop On File “stopon==“ Option 35
Sleep “*sleep=“ Options 35
Show Block Tag “showbtag“ Option 35
Show File System LBA “showfslba“ Option 36
Show File System Map “showfsmap“ Option 36
Show Time “showtime=“ Option 36
Show Verify Flags “showvflags=“ Option 36
Threads “threads=“ Option 37
Trigger Type “trigger=“ Option 37
Trigger Action “trigger_action=“ Option 37
Trigger On “trigger_on=“ Option 38
Terminal Speed “speed=“ Option 38
Terminal Read Timeout “timeout=“ Option 38
Terminal Read Minimum “ttymin=“ Option 39
Multiple Volumes “volumes=“ Option 39
Multi-Volume Records “vrecords=“ Option 39
Other Commands 39
I/O Behavior (“behavior=) Option 39
Force Corruption (“corrupt_*=) Options 40
Job Control Options 40
File Locking Options 41
Define Workload (“define”) Option 41
Workload (“workload=”) Option 41
File System Full (“fsfree_*”) Options 41
Retry Error (“retry_*”) Options 42
SCSI Specific Options 42
Enable “enable=“ and Disable “disable=“ Options 42
  POSIX Asynchronous I/O “*aio*” Flag 43
  Asynchronous Job “async” Flag 43
  Reporting Close Errors “*cerror*” Flag 43

Block Tag “btags” Flag 43
Data Compare “*compare*” Flag 44
Extra Block Tag Compare “x*compare*” Flag 44
Core Dump on Errors “*coredump*” Flag 44
Delete Error Log File “deleteerrorlog” Flag 44
Dump Block Tags “dump_btags” Flag 44
Dump Context “dump_context” Flag 44
Report Errors “errors” Flag 45
Report Extended Errors “xerrors” Flag 45
File Per Thread “fileperthread” Flag 45
File File Once “fill_once” Flag 45
Always Fill Files “fill_always” Flag 45
File System Map Control Flag “fsmap” Flag 45
File System Trim “fsmap” Flag 45
File System Increment “fsincr” Flag 46
Log File Trailer “fsincr” Flag 46
Force Corruption “force-corruption” Flag 46
Image Mode Copy “image” Flag 46
I/O Lock “iolock” Flag 46
Lock Files “lockfiles” Flag 47
Millisecond Delay “msecdelay” Flag 47
Second Delays “secsdelay” Flag 47
Mount Lookup “mount_lookup” Flag 47
Pipe Mode Control “pipes” Flag 47
Poison Read Buffer “poison” Flag 47
Image Mode Copy “job_stats” Flag 48
Total Statistics “total_stats” Flag 48
Re-Read After Read-After-Write “reread” Flag 48
Restart File System Full “resfsfull” Flag 48
Retry Session Disconnects “retrydisc” Flag 48
Retry Warning “retrywarn” Flag 49
Save Corrupted Data “savecorrupted” Flag 49
Script Verify “scriptverify” Flag 49
Hangup Signal Control “sighup” Flag 49
Sparse File Attribute “sparse” Flag 49
Pre-Allocate Without Sparse “prealloc” Flag 49
SCSI Operation “scsi*” Flags 50
Stop Immediate w/Stop File “stopimmed” Flag 50
Terminate on Signals “terminate_on_signals” Flag 50
Trigger Control “trig*” Flags 50
UUID Dashes “uuid_dashes” Flag 50
Diagnostic Logging “*diag*” Flag 51
Debug Output “*debug*” Flag 51
Verbose Debug Output “*Debug*” Flag 51
Other Debug Output “**debug*” Flags 51
Delete Per Pass “*deleteperpass*” Flag 51

Dump Data Buffer “*dump*” Flag 52
Tape EEI Reporting “*eei*” Flag 52
Flush Terminal I/O Queues “*flush*” Flag 52
History Dumping “*hdump*” Flag 52
History Timing “*htiming*” Flag 53
Log File Header “*header*” Flag 53
Loop On Error “*looponerror*” Flag 53
Logical Block Data Mode “*lbdata*” Flag 53
Enable Loopback Mode “*loopback*” Flag 53
Microsecond Delays “*microdelay*” Flag 54
Memory Mapped I/O “*mmap*” Flag 54
Test Modem Lines “*modem*” Flag 55
Multiple Volumes “*multi*” Flag 55
No I/O Progress “*noprog*” Flag 55
Prefill “*prefill*” Flag 55
Control Per Pass Statistics “*pstats*” Flag 56
Read After Write “*raw*” Flag 56
Tape Reset Handling “*resets*” Flag 56
Retry Data Corruptions “*retryDC”* Flag 56
Control Program Statistics “*stats*” Flag 56
Table(sysinfo) timing “*table*” Flag 57
System Log “syslog” Flag 57
Timestamp Blocks “*timestamp*” Flag 57
Unique Pattern “*unqiue*” Flag 57
Verbose Output “*verbose*” Flag 57
Verify Data “verify” Flag 58
Program Delays 58
Close File “*close_delay*=“ Delay 58
Delete File “*delete_delay*=“ Delay 58
End of Test “*end_delay*=“ Delay 58
Open “open_delay=“ Delay 58
Read Record “*rread_delay*=“ Delay 59
Start Test “*start_delay*=“ Delay 59
Terminate Thread “*term_delay*=“ Delay 59
Terminate Wait “*term_wait*=“ Time 59
Verify “verify_*delay*=“ Delay 59
Write Record “*write_delay*=“ Delay 59
Numeric Input Parameters 60
Time Input Parameters 60
Future Enhancements 60
Final Comments 61
Appendix A *dt* Help Text 62
Appendix B Test Strategy 71
Appendix C *dt* Workloads 73
Reviewing Workloads 73
Executing Workloads 74

Defining Workloads 75
Appendix D Block Tags 77
Appendix E *IOT Pattern* 78
Appendix F *File System Map* 79
Appendix G *Trigger Script* 80
Appendix H S3 Cloud *Storage* 82
Copy of s3t.sh Script 83
Copy of test-s3t.sh Script 84
Example s3t.sh Log File 85

Data **T**est (dt) Program

Overview

dt is a generic data test program used to verify the proper operation of peripherals and I/O sub-systems, and for obtaining performance information. Since verification of data is performed, *dt* can be thought of as a generic diagnostic tool.

Although the original design goals of being a generic test tool were accomplished, it quickly become evident that device specific tests, such as terminals, and different programming interfaces such as memory mapped files and POSIX asynchronous I/O API's were necessary. Therefore, special options were added to enable these test modes and to specify necessary test parameters.

dt command lines are similar to the *dd* program, which is popular on most UNIX systems. *dt* contains numerous options to provide user control of most test parameters so customized tests can be written easily and quickly by specifying simple command line options. Since the exit status of the program always reflects the completion status of a test; scripts can easily detect failures to perform automatic regression tests.

dt has been used to successfully test disks, tapes, serial lines, parallel lines, pipes & FIFO's, memory mapped files, and POSIX Asynchronous I/O. In fact, *dt* can be used with any device that supports the standard open, read, write, and close system calls. Special support is necessary for some devices, such as serial lines, for setting up the speed, parity, data bits, etc, but *dt*'s design provides easy addition of this setup.

Most tests can be initiated by a simple *dt* command line, and lots of I/O can be initiated quickly using multiple processes and/or POSIX AIO, for those operating systems supporting AIO. More complex tests are normally initiated by writing shell scripts and using *dt* in conjunction with other tools, such as *scu* (SCSI Command Utility). Several shell scripts for testing disks, tapes, and serial lines are also supplied with this kit which can be used as templates for developing other specialized test scripts.

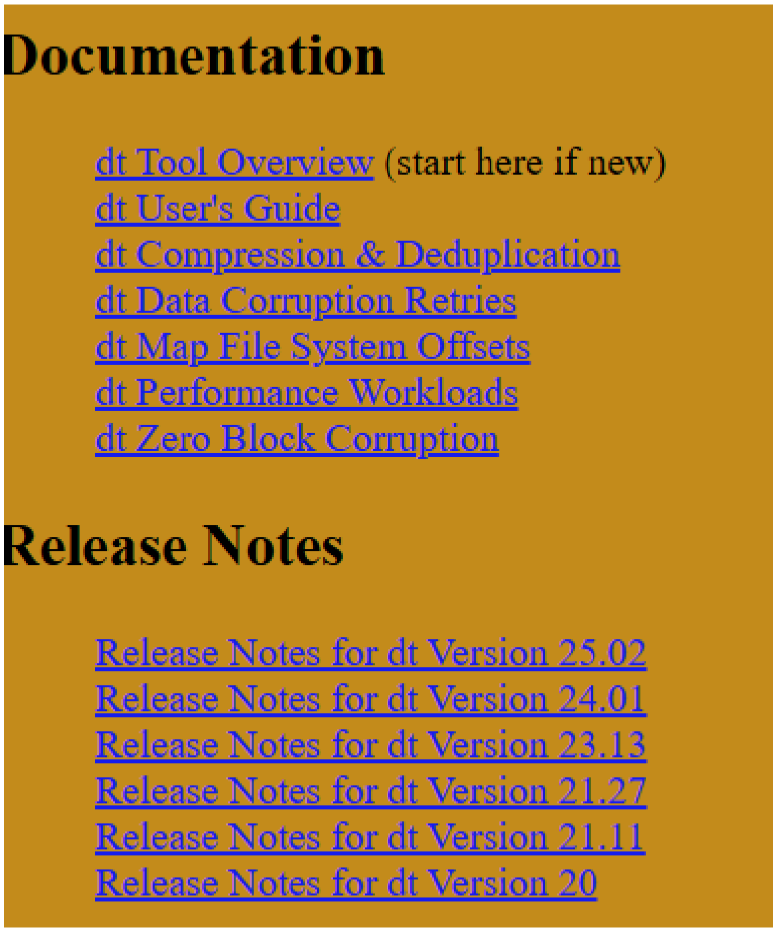
Specific system features are now being added to *dt* so more extensive testing can be accomplished. The program has been restructured to allow easy inclusion of new device specific tests by dispatching to test functions through a function lookup table. This table gets set up automatically, based on options enabled, or via the device type "*dtype=*" option.

Please Note: While the original *dt* supported multiple device types, such as serial and parallel lines, and magnetic tapes, the latest version has mainly been concentrated on random access storage devices, such as disks and file systems. Therefore, if this alternate device support is enabled via compile options, there’s no guarantee it works today.

WARNING: *dt* does not perform any sanity checking of the output device specified. This means if you are running as root on Unix and you specify a raw disk device, dt will overwrite existing file systems, so please use care! I HATE TO ADMIT, I’VE DONE THIS MYSELF!

Additional Documentation

After downloading the *dt* distribution from Github, please open html/dt.html to access this documentation list: (below is an image, so *not* clickable)



Documentation

dt Tool Overview (start here if new)
dt User's Guide
dt Compression & Deduplication
dt Data Corruption Retries
dt Map File System Offsets
dt Performance Workloads
dt Zero Block Corruption

Release Notes

Release Notes for dt Version 25.02
Release Notes for dt Version 24.01
Release Notes for dt Version 23.13
Release Notes for dt Version 21.27
Release Notes for dt Version 21.11
Release Notes for dt Version 20

New Feature Summary

The last update for this user guide was for *dt* version 17.38, while updates are for version 25.05. This is a summary of the latest features for this updated user guide:

- Multiple Threads vs. multiple processes
- CLI, Interactive, Pipe, and Script modes
- Block Tags
- Extended Error Reporting
- Improved Triggers
- Corruption Analysis (with IOT pattern)
- Asynchronous Jobs w/Job Control
- SCSI I/O
- NVME I/O
- Multiple I/O Behaviors
- Predefined Workloads

- Storage Outage Support
- Create Corruption Options
- Cloud Object Storage Testing

There are many other features, but the above are the major ones. 😊

Operating Systems Supported

dt is conditionalized to run on AIX, HP-UX, Solaris, ULTRIX, OSF, OSX, QNX, SCO Unixware, Windows, and Linux operating systems. Porting is fairly simple for OS's with POSIX API's.

Program Options

This section describes program options and special notes related to each. The *dt* help file provides a summary of the options, and the default value of most options. The *dt* help summary is shown in Appendix A.

Input File "*if*=" Option

This option specifies the input file to open for reads. The device is opened read-only so devices which only permit or support read access, e.g., parallel input devices, can be opened successfully.

Special Notes:

- Data read is automatically verified with the default data pattern, unless you disable this action via the "disable=compare" option.
- Extra pad bytes of sizeof(int), are allocated at the end of data buffers, initialized with the inverted data pattern, and then verified after each read request to ensure the end of data buffers didn't get overwritten by file system and/or device drivers. This extra check has found problems with flushing DMA FIFO's on several machines.
- This option accepts a comma separated list, each will be created in its' own job.

Syntax:

if=filename	The input file to read.

Output File "*of*=" Option

This option specifies the output file to open for writes. After the writing portion of the test, the device is closed (to reposition to start of file or to rewind the tape), re-opened, and then a read verification pass is performed. If you wish to prevent the read verify pass, you must specify the "*disable=verify*" option.

Special Notes:

- Terminal devices are **not** closed between passes so previously set terminal characteristics don't get reset. This also caused a race condition when doing loopback testing with two processes.
- When testing terminal (serial) devices, modem control is disabled (via setting CLOCAL) to prevent tests from hanging. If the "*enable=modem*" option is specified, then CLOCAL is reset, hangup on close HUPCL is set, and testing will not proceed until carrier or DSR

is detected. This code is not fully tested, but this description accurately describes the code.

- At the present time, tapes are rewound by closing the device, so you *must* specify the rewind device during testing if the read verify pass is being performed. This restriction will probably change in the next release since magtape control commands will be supported (tape specific tests as well).
- A special check is made for the /dev/ prefix, and if located, the O_CREAT open flag is cleared to prevent accidentally creating files in this directory when not specifying the correct device name (very easy to do when running tests as super-user 'root').
- When writing to raw disks on Tru64 UNIX, if the disk was previously labeled, you must issue the "*disklabel -z*" command to destroy the label block or else you cannot write to this area of this disk (block 0). Failure to do this results in the error "Read-only file system" (errno=EROFS) being returned on write requests.
- This option accepts a comma separated list, each will be created in its' own job.

Syntax:

of=filename	The output file to write.

SCSI Device "sdsf=" Option

This option overrides the normal input/output device to acquire SCSI device information. This is useful when testing virtual LUNs on say VMware, by providing a physical LUN when exposed. This is also the SCSI device used for sending SCSI triggers, if the trigger device is not specified.

Syntax:

sdsf=filename	The SCSI device special file.

SCSI Trigger "tdsf=" Option

This option specifies an alternate SCSI device to send SCSI trigger to. This is useful for example when testing say a FCP device, but when all paths disappear, an iSCSI device can be used to send to send the trigger command to the array controller, which can then take action.

Syntax:

tdsf=filename	The SCSI trigger device file.

Pattern File "*pf=*" Option

This option specifies a pattern file to use for the data pattern during testing. This option overrides the "*pattern=*" option and allows you to specify specialized patterns. The only restriction to this option is that the entire file *must* fit in memory. A buffer is allocated to read the entire pattern file into memory before testing starts so performance is not affected by reading the pattern file.

Syntax:

pf=filename	The data pattern file to use.

Block Size "*bs*=" Option

This option specifies the block size, in bytes, to use during testing. At the present time, this option sets both the input and output block sizes. At the time I originally wrote this program, I didn't have the need for separate block sizes, but this may change in a future release where I'll add back the "*ibs*=" and "*obs*=" options available with *dd*.

Special Notes:

- When enabling variable length records via the "*min*=" option, this also sets the maximum record size to be written/read.
- For memory mapped files, the block size *must* be a multiple of the system dependent page size (normally 4k or 8k bytes).

Syntax:

bs=value	The block size to read/write.
or bs=random	Selects random size between 512 and 1m.

Log File "*log[atu]*=" Options

This option specifies the log file to redirect all program output to. This is done by re-opening the standard error stream (stderr) to the specified log file. Since all output from *dt* is directed to stderr, library functions such as perror() also write to this log file.

Special Notes:

- A separate buffer is allocated for the stderr stream, and this stream is set buffered so timing isn't affected by program output.
- When starting multiple processes via the "*procs*=" option, all output is directed to the same log file. The output from each process is identified by the process ID (PID) as part of the message (errors & statistics).
- loga=filename will append to an existing log file.
- logt=filename will truncate the existing log file.
- logu=filename will create unique log files with multiple processes (w/pid).

Syntax:

log[atu]=filename	The log file name to write.

Special format keywords are now expanded when part of the log file name, so unique names can be created for each test, where they make sense:

```
Common Format Control Keywords:
  %array   = The array name or management IP.
  %bufmode = The file buffer mode.  %dfs    = The directory separator ('/')
  %dsf     = The device name.       %device = The device path.
  %sdsf    = The SCSI device name.  %tdsf   = The trigger device name.
  %file    = The file name.         %devnum = The device number.
  %host    = The host name.         %user   = The user name.
  %job     = The job ID.            %tag    = The job tag.
  %jlog    = The job log.           %tlog   = The Thread log.
  %tid     = The thread ID.         %thread = The thread number.
  %pid     = The process ID.        %prog   = The program name.
```

```
%ymd     = The year,month,day.    %hms     = The hour,day,seconds.
%date    = The date string.       %et      = The elapsed time.
%tod     = The time of day.       %etod    = Elapsed time of day.
%secs    = Seconds since start.   %seq     = The sequence number.
%script  = The script file name.  %tmpdir = The temporary directory.
%uuid    = The UUID string.       %workload = The workload name.
%month   = The month of the year. %day     = The day of the month.
%year    = The four digit year.   %hour    = The hour of the day.
%minutes = The minutes of hour.   %seconds= The seconds of minute.
%nate    = The NATE log prefix.   %nos     = The Nimble log prefix.
```

String 'gtod' = "%tod (%etod) %prog (j:%job t:%thread): "

Example: log=dt_%host_%user_%iodir_%iotype-%uuid.log
logprefix="%seq %ymd %hms %et %prog (j:%job t:%thread): "

POSIX Asynchronous I/O "*aios*=" Option

This option enables and controls the number of POSIX Asynchronous I/O (AIO) requests.

Special Notes:

- The default is to queue up to 8 AIO requests.
- The system limit for AIO on Tru64 UNIX is dynamic and can be queried by using the "*sysconfig -q rt*" command.
- You can use the "*enable=aio*" option to enable AIO and use the default request limit.
- AIO is only supported for character devices and is disabled for terminals. On Tru64 UNIX, you can alter the Makefile and link against libaio.a, which allows AIO with any device/file by mimic'ing AIO using POSIX threads.
- AIO requests can **not** be cancelled on Tru64 UNIX, so queuing many requests to 1/2" tape devices will probably result in running off the end of the tape reel. This is not a problem for cartridge tapes.

Syntax:

```
aios=value       Set number of AIO's to queue.
```

Keepalive Alarm Time "*alarm*=" Option

Keepalive Message "**keepalive*=" Options

These options control a user defined message that will be emitted during the test. The user defines how often to display the keepalive message, via the "*alarm=time*" option, and the format of the message(s), via the "**keepalive=string*" options. The normal "*keepalive=*" option defines the script emitted during the test, while "*pkeepalive=*" is the per pass message string, and "*tkeepalive=*" is the totals message string (overriding what *dt* normally displays).

Syntax:

```
alarm=time       The keepalive alarm time.
keepalive=string The keepalive message string.
keepalivet=time  The keepalive message frequency.
pkeepalive=str   The pass keepalive msg string.
```

```
        tkeepalive=str   The totals keepalive msg string.
```

Keepalive Message Format Control

The keepalive string is free format like a printf(), with the following format control strings:

```
    Keepalive Format Control:
        %b = The bytes read or written.  %B = Total bytes read and written.
        %c = Record count for this pass. %C = Total records for this test.
        %d = The device name.            %D = The real device name.
        %e = The number of errors.       %E = The error limit.
        %f = The files read or written.  %F = Total files read and written.
        %h = The host name.              %H = The full host name.
        %k = The kilobytes this pass.    %K = Total kilobytes for this test.
        %l = Blocks read or written.     %L = Total blocks read and written.
        %m = The megabytes this pass.    %M = Total megabytes for this test.
        %p = The pass count.             %P = The pass limit.
        %r = Records read this pass.     %R = Total records read this test.
        %s = The seconds this pass.      %S = The total seconds this test.
        %t = The pass elapsed time.      %T = The total elapsed time.
        %i = The I/O mode (read/write)   %u = The user (login) name.
        %w = Records written this pass.  %W = Total records written this test.

    Performance Keywords:
        %bps  = The bytes per second.    %lbps = Logical blocks per second.
        %kbps = Kilobytes per second.    %mbps = The megabytes per second.
        %iops = The I/O's per second.    %spio = The seconds per I/O.

      Lowercase means per pass stats, while uppercase means total stats.

      Default: %d Stats: mode %i, blocks %l, %m Mbytes, pass %p/%P, elapsed %t
                     or if pass statistics summary is disabled:
                %d Stats: mode %i, blocks %L, %M Mbytes, pass %p/%P, elapsed %T
```

Here's an example used by Hazards' *diskdt* process:

```
  keepalive="count = %C; e = %e; t = %S; IOpS = %IOPS; SpIO = %SPIO"
  tkeepalive="STAT +RawMbytes %MBPS +RawReads %R +RawWrites %W";
```

Other Format Control Keywords

```
   I/O Keywords:
     %iodir = The I/O direction.       %iotype = The I/O type.
     %lba = The current logical block. %offset = The current file offset.
     %elba = The error logical block.  %eoffset = The error file offset.
     %bufmode = The file buffer mode.  %status = The thread exit status.

    Job Control Keywords:
     %job  = The job ID.               %tag    = The job tag.
     %tid  = The thread ID.            %thread = The thread number.

    Misc Keywords:
      %keepalivet = The keepalive time.
```

Buffer Alignment "*align*=" Option

This option controls the alignment of the normally page aligned data buffer allocated. This option is often useful for testing certain DMA boundary conditions that are not easily reproduced otherwise. The rotate option automatically adjusts the data buffer pointer by (0, 1, 2, 3, ...) for each I/O request to ensure various boundaries are fully tested.

```
 Syntax:
    align=offset     Set offset within page aligned buffer.
or  align=rotate     Rotate data address through sizeof(ptr).
```

File Disposition "*dispose*=" Option

This option controls the disposition of test files created on file systems. By default, the test file created is deleted before exiting, but sometimes you may wish to keep this file for further examination, for use as a pattern file, or simply for the read verify pass of another test (e.g., reading the file via memory map API).

Syntax:
dispose=mode   Set file dispose to: {delete, keep, or keeponerror}.

Dump Data Limit "*dlimit*=" Option

This option allows you to specify the dump data limit used when data compare errors occur. The default dump data limit is 64 bytes.

Syntax:
dlimit=value   Sets the data dump limit to value.

Data Corruption Analysis Options

Several options have been added to improve data corruption analysis when using the IOT data pattern, the preferred pattern for maximum data validation.

Options Added:

```
boff=string       Set the buffer offsets to: dec or hex.(Default: hex}
dfmt=string       Set the data format to: byte or word. (Default: byte}
maxbad=value      Set maximum bad blocks to display.    (default is 1)
enable=dumpall    Dump all blocks (good and bad).       (Default: disabled)
```

The file offset and block range of each corruption and good block is summarized, and the IOT data is analyzed to determine if it's from a different pass, indicating stale data. When the IOT pattern is selected, the data format is set to "dfmt=word", since IOT data is packed in 32-bit word values.

Dump Format "*dfmt*=" Option

This option controls the dump format used with corruption analysis performed on IOT data.

Syntax:
dfmt=string   Set the data format to: byte or word (Default: word)

Buffer Offset "boff=" Option

This option controls the radix of the buffer offsets displayed with IOT corruption analysis.

```
Syntax:
   boff=string      Set the buffer offsets to: dec or hex (Default: hex)
```

Max Bad Blocks "maxbad=" Option
This option controls how many bad blocks to display during IOT corruption analysis. By default, all bad blocks will be displayed. The optional enable=dumpall will also dump good data blocks.

```
Syntax:
   maxbad=value     Set maximum bad blocks to display.
```

Device Size "*dsize*=" Option
This option allows you to specify the device block size used. On Tru64 Unix, the device block size is obatined automatically by an OS specific IOCTL. For all other systems, random access devices default to 512 byte blocks. You'll likely use this option with C/DVD's, since their default block size to 2048 bytes per block.

```
Syntax:
   dsize=value      Set the device block (sector) size.
```

Device Type "*dtype*=" Option

Input Device Type "*idtype*=" Option

Output Device Type "*odtype*=" Option
These options provide a method to inform *dt* of the type of device test to be performed. Without this knowledge, only generic testing is possible.

Special Notes:

- On Tru64 UNIX systems, these options are not necessary, since this information is obtained via the DECIOCGET or DEVGETINFO IOCTL's.
- Although the program accepts a large number of device types, as shown below, specific tests only exists for "disk", "tape", "fifo", and "terminal" device types. Others may be added in the future.
- In the case of "disk" device type, *dt* reports the relative block number when read, write, or data compare errors occur.
- Also for "disk" devices, *dt* will automatically determine the disk capacity if a data or record limit is not specified. This is done via a series of seek/read requests.
- On each operating system supported, string compares are done on well known device names to automatically select the device type. For example on QNX, "/dev/hd" for disk, "/dev/tp" for tapes, and "/dev/ser" for serial lines.
- The device type gets displayed in the total statictics.

```
Syntax:
    dtype=string    Sets the device type.
    idtype=string   Sets the input device type.
    odtype=string   Sets the output device type.

 The Valid Device Types Are:
    audio     comm      disk      graphics  memory
    mouse     network   fifo      pipe      printer
    processor socket    special   streams   tape
    terminal  unknown
```

Note: Although *dt* does not provide specific test support for each of the devices shown above, its' design makes it easy to add new device specific tests. Specific support exists for disk, fifo, pipe, tape, and terminals. Support for "ptys" may be added in the future as well.

Error Limit "*errors*=" Option

This option controls the maximum number of errors tolerated before the program exits.

Special Notes:

- The default error limit is 1.
- All errors have a time stamp associated with them, which are useful for characterizing intermittent error conditions.
- The error limit is adjusted for read, write, or data compare failures. This limit is not enforced when flushing data, or for certain AIO wait operations which are considered non-fatal (perhaps this will change).
- A future release may support an "*onerr=*" option to control the action of errors (e.g., loop, ignore (continue), or exit).

Syntax:

errors=value   The number of errors to tolerate.

File System Test Options

This is a summary of the directory/file options added:

```
           dir= (top level directory)
    1 to files= (for multiple files)
    1 to sdirs= (for multiple subdirectories)
    1 to depth= (subdirectory depth, nested subdirs)
  enable=fdebug (enable file system debugging only)
    dirp=string  The directory prefix for subdirs.
 maxdata=value  The maximum data limit (all files).
maxfiles=value  The maximum files for all directories.
bufmodes={buffered,unbuffered,cachereads,cachewrites}
                Set one or more buffering modes (Default: none)
enable/disable file system caching: (or use new format above)
    readcache        Read cache control.    (Default: enabled)
    writecache       Write cache control.   (Default: enabled)
    deleteperpass    Delete files per pass. (Default: disabled)


The files= option creates many files in each directory.
Multiple procs= creates multiple top level directories w/PID.
The file system catching options enable or disable direct I/O.
```

File System Buffering Modes "bufmodes=" Option

This option allow you to control file system behavior. When multiple buffering modes are specified, dt round-robins the modes during each pass.

```
Syntax:
    bufmodes={buffered,unbuffered,cachereads,cachewrites}
                Set one or more buffering modes (Default: none)
```

This is an alternative to using the enable/disable{readcache,writecache} option.

Directory Path "dir=" Option

This option specifies the top-level directory when creating multiple files/directories.

Special Notes:

- When specifying a directory path, the input/output file options do not require a path. The file name is appended to the directory path appropriately for Unix vs. Windows.
- If the last part of the directory path does not exist, dt creates the directory (please use "mkdir -p" to create all paths). For example, dir="/var/tmp/dtdir" will create "dtdir" in /var/tmp, if it doesn't exist.
- When used in conjunction with multiple processes, the process ID is appended to the directory name, and the directory is created.
- When dispose=delete, after deleting files, the directory created is also deleted.
- The directory path, previously required via the dir=path option, is now optional, if you of= or if= options specify a directory path
- dt will detect the path, and automatically setup the directory path for you, making it easier to add multiple files/directories to existing scripts.

```
Before: dir=/var/tmp of=dt.data (two parameters)
   Now: of=/var/tmp/dt.data gets broken into dir=/var/tmp of=dt.data
```

```
Syntax:
   dir=dirpath      The directory path for files.
```

Directory Prefix "dirp=" Option

This option specifies a prefix to use with subdirectory names, overriding the default name of “dN”, where ‘N’ is the subdirectory name, e.g. “d0”.

```
Syntax:
   dirp=string      The directory prefix for subdirs.
```

Number of Subdirectories "sdirs=" Option

This option specifies the number of subdirectories to create.

Special Notes:

- sdirs= is the subdirectory width, while depth= is the height which controls the number of nested subdirectories under each top level subdirectory.

- The default directory name prefix is "d", short so one can nest deeper without exceeding the path limit (Linix is 4096). But, dirprefix= allows you to override the direcotry prefix or set to "" for no prefix.
- Like the multiple files option, when exiting dt will delete all subdirectory files and directories, assuming dispose=delete.
- When sending dt signals, please use SIGINT or SIGTERM, so cleanup is possible. If you kill dt via SIGKILL (-9), then dt won't get a chance to do its' cleanup! Also remember, if dt created lots of files and directories, it may take awhile.
- File state is maintained, so file system full conditions read the same number of files and data as was written during the write pass

.

Syntax:

sdirs=value	The number of subdirectories.

Subdirectory Depth "depth=" Option

This option specifies the depth of subdirectories in each subdirectory created.

Syntax:

depth=value	The subdirectory depth.

File Limit "*files*=" Option

This option controls the number of disk or tape files to process.

Special Notes for Tapes:

- During the write pass, a tape file mark is written after each file. After all files are written, 1 or 2 file marks will be written automatically by the tape driver when the device is closed.
- During reads, each file is expected to be terminated by a file mark and read() system calls are expected to return a value of 0 denoting the end of file. When reading past all tapes files, an errno of ENOSPC is expected to flag the end of media condition.
- Writing tape file marks is currently not supported on the QNX Operating System. The release I currently have does not support the mtio commands, and unfortunately the POSIX standard does **not** define this interface (the mtio interface appears to be a UNIX specific standard). Multiple tape files can still be read on QNX systems however.

Special Notes for Disk Files:

- If dispose=delete, dt deletes all files when exiting.
- If dispose=keeponerror, signals are considered errors so files are kept.
- If a file system full condition is encountered, the write pass stops and the read pass will read as many files as were written. Subsequent passes will either overwrite previous file data, or you can include the truncate flag (oflags=trunc) to free space and loop on file system fills.
- When using a prefix string with the device name in its' format (%d), the prefix string is updated with each new file name to create uniqueness in each data block.
- When using the IOT data pattern, the file number is factored into the IOT pattern to generate unique data for each file, creating a different seed for each block.

- When dt's internal data patterns are used, the pass count and file number are used to cycle through dt's internal data patterns, again to create unique data per file.
- A postfix of "-NNNNNNNN" is added to each file name for uniqueness.
- Reference disable={readcache,writecache} to control file system caching.

Syntax:
 files=value Set number of disk/tape files to process.

File Frequency Flush "ffreq=" Option

This option specifies how frequently to flush file system data in records. Normally the flush only happens are the end of the write pass, unless disable=fsync is specified.

This translates into a Unix fsync() or Windows FlushFileBuffers() to push data from the host buffer cache to the underlying physical storage. This may be helpful during file system writing, to detect write failures that are otherwise unknown to dt due to their async nature. When write failures are missed, dt reports a data corruption during its' read/compare pass. This option may be used in lieu to Direct I/O, which does catch write failures, when buffered I/O is preferred

Syntax:
 ffreq=value The frequency (in records) to flush buffers.

Maximum Data "maxdata=" Option

The maximum data created by all files can be specified via the maxdata= option. This is useful if you know the file system space, and wish to either divide this space between multiple test clients (NAS testing), or prevent dt from encountering a file system full condition (which is handled properly now).

Syntax:
 maxdata=value The maximum data limit (all files).

Maximum Data Percentage "maxdatap=" Option

This option specifies the maximum data percentage, to avoid using the full capacity.

Syntax:
 maxdatap=value Set capacity by percentage (range: 0-100).

For example:
 maxdatap=50 sets the capacity to 50% of the full capacity of the disk.

Maximum Files "maxfiles=" Option

The maximum files created by in all directories can be specified via the maxfiles= option. This is useful if you don't wish to do the math to figure out how many files per directory and/or subdirectory are required to say create 1 million files.

Syntax:
 maxfiles=value The maximum files for all directories.

File Plush Frequency “ffreq=” Option
This option specifies how frequently flush file system data when caching is enabled.

Syntax:

ffreq=value	The frequency (in records) to flush buffers.

File System Trim Frquency “fstrim_freq=” Option
This option controls how often (in files) to perform file system trim operations.
Note: This is only supported on Windows today.

Syntax:

fstrim_freq=value	The file system trim frequency (in files).

Write Fill Pattern “fill_pattern=” Option
This option specifies the fill pattern to use when performing write only operations.

Syntax:

fill_pattern=value	The write fill pattern (32 bit hex).

Read Prefill Pattern “prefill_pattern=”) Option
This option specifies a prefill pattern to initialize the first 4 bytes of each disk sized buffer. Generally this is not required, but there have been times when the read buffer may not get overwritten and will contain data from the previous read.

Note: This option is automatically enabled by enable=prefill or enable=poison is flags.

Syntax:

prefill_pattern=value	The read prefill pattern (32 bit hex).

Terminal Flow Control "*flow*=" Option
This option specifies the terminal flow control to be used during testing.
Please Note: Serial line support is disabled during compilation. Unknow to work today.

Special Notes:

- The default flow control is "xon_xoff".
- When using XON/XOFF flow control, you must make sure these byte codes (Ctrl/Q = XON = '\021', Ctrl/S = XOFF = '\023), since the program does not filter these out automatically. Also be aware of terminal servers (e.g., LAT), or modems (e.g., DF296) which may eat these characters.
- Some serial lines do **not** support clear-to-send (CTS) or request-to-send (RTS) modem signals. For example on Alpha Flamingo machines, only one port (/dev/tty00) supports full modem control, while the alternate console port (/dev/tty01) does not. Therefore, if running loopback between both ports, you can not use *cts_rts* flow control, the test will hang waiting for these signals to transition (at least, I think this is the case).

Syntax:
```
  flow=type       Set flow to: none, cts_rts, or xon_xoff.
```

History "history=" Option

This option sets the number of I/O history entries to record. During failures, the history is dumped, which can be helpful when troubleshooting failures.

Syntax:
```
  history=value    Set the number of history request entries.
```

History Buffer "history_bufs=" Option

This option controls how many history data buffers are allocated for record data. Usually only one history data buffer is allocated which saves history data (hdsize=) bytes per record. Using this option, data from multiple record blocks can be saved, for example every 512 bytes or 4k.

Syntax:
```
  history_bufs=value Set the history data buffers (per request).(or hbufs)
```

History Block Data Size "history_bsize=" Option

This option defines the block data size increment. By default this will be the file or disk block size, but using a protocol specific block size is often desirable, like 2k frame size, etc.

Syntax:
```
  history_bsize=value    Set the history block data size increment. (or hbsize)
```

History Data Size "hdsize=" Option

When I/O history is enabled, this option controls how many data bytes are saved for each I/O.

Syntax:
```
     hdsize=value     Set the history data size (bytes to save).
                      Default hdsize=32 (set to 0 to disable copy)
```

Record Increment "incr=" Option

This option controls the bytes incremented when testing variable length records. After each record, this increment value (default 1), is added to the last record size (starting at "*min=*", up to the maximum record size "*max=*").

Special Notes:

- If variable length record testing is enabled on fixed block disks and this option is omitted, then "*incr=*" defaults to 512 bytes.

```
  Syntax:
     incr=value        Set number of record bytes to increment.
  or incr=variable     Enables variable I/O request sizes.
```

I/O Per Second “iops=” Option

This option specifies the I/O’s per second. This floating point value is applied per thread.

```
Syntax:
   iops=value       Set I/O per second (this is per thread).

Example:
   iops=.25
```

I/O Direction "*iodir*=" Option

This option allows you to control the I/O direction with random access devices. The default direction is forward. Specifying “vary” will randomly choose the direction.

Syntax:
Set I/O direction to: {forward, reverse, or vary}.

I/O Mode "*iomode*=" Option

This option controls the I/O mode used, either copy, test, or verify modes. The copy option was added to do a byte for byte copy between devices, while skipping bad blocks and keeping file offsets on both disks in sync. I've used this option to (mostly) recover my system disk which developed bad blocks which could not be re-assigned. A verify operation automatically occurs after the copy, which is really handy for unreliable storage, such as (ancient) diskettes.

Syntax:
iomode=mode   Set I/O mode to: {copy, mirror, test, or verify}.

IOT Pass "*iotpass*=" Option

This option is used to specify the IOT pass number. When multiple passes occur, dt factors in the pass count to generate unique data during each pass. For example, the IOT seed is normally 0x01010101, and will be multiplied by the pass specified, useful for re-reading previously written IOT data patterns.

Syntax:
iotpass=value   Set the IOT pattern for specified pass.

IOT Seed "*iotseed*=" Option

This option is used to specify the last IOT pattern seed dt used. When multiple passes occur, dt now factors in the pass count to generate unique data during each pass. For example, the IOT seed is normally 0x01010101, but this is now multiplied by the pass count for uniqueness.

Syntax:
iotseed=value   Set the IOT pattern block seed value.

I/O Tune File "*iotune*=" Option
This option is used to change I/O tuning parameters to dynamically control I/O's per second. This is generally used in conjunction with automation which is monitoring the storage under test to specifically avoid overdriving with too many I/O's, possibly from multiple test hosts.

The format of this file are options like you'd specify on the command line, for example:
read_delay=value, write_delay=value, or enable/disable=*{debug flags}*

This provides a method to dynamically control delays and debug flags.

I/O Type "*iotype*=" Option
This option controls the type of I/O performed, either random or sequential. The default is to do sequential I/O. Specifying "vary" will randomly choose the I/O type.

Syntax:

iotype=type	Set I/O type to: {random, sequential, or vary}.

The seeks are limited to the data limited specified or calculated from other options on the *dt* command line. If data limits are not specified, seeks are limited to the size of existing files, or to the entire media for disk devices (calculated automatically by *dt*). If the data limits exceed the capacity of the media/partition/file under test, a premature end-of-file will be encountered on reads or writes, but this is treated as a warning (expected), and not as an error.

Minimum Record Size "*min*=" Option
This option controls the minimum record size to start at when testing variable length records.

Special Notes:

- By default, *dt* tests using fixed length records of block size "*bs*=" bytes.
- This option, in conjuntion with the "*max*=" and "*incr*=" control variable length record sizes.
- If variable length record testing is enabled on fixed block disks and this option is omitted, then "*min*=" defaults to 512 bytes.

Syntax:

min=value	Set the minimum record size to transfer.

Maxmimum Record Size "*max*=" Option
The option controls the maximum record size during variable length record testing.

Special Notes:

- If the "*min*=" option is specified, and this option is omitted, then the maximum record size is set to the block size "*bs*=".
- This option, in conjuntion with the "*min*=" and "*incr*=" control variable length record sizes.

Syntax:

max=value Set the maximum record size to transfer.

Logical Block Address “*lba*=“ Option

This option sets the starting logical block address used with the “*lbdata*” option. When specified, the logical block data (*enable=lbdata*) option is automatically enabled.

Syntax:
lba=value Set starting block used w/lbdata option.

Special Notes:

- Please do not confuse this option with the disks' real logical block address. See *dt*'s “*seek=*“ or “*position=*“ options to set the starting file position.
- Also note that *dt* doesn't know about disk partitions, so any position specified is relative to the start of the partition used.

Logical Block Size “*lbs*=“ Option

This option sets the starting logical block size used with the *lbdata* option. When specified, the logical block data (*enable=lbdata*) option is automatically enabled.

Syntax:
lbs=value Set logical block size for lbdata option.

Data Limit “*limit*=“ Option

This option specifies the number of data bytes to transfer during each write and/or read pass for the test.

Special Notes:

- You must specify either a data limit, record limit, or files limit to initiate a test, unless the device type is "*disk*", in which case *dt* will automatically determine the disk capacity.
- When specifying a runtime via the “*runtime=*“ option, the data limit controls how many bytes to process for each pass (write and/or read pass).
- If you specify a infinite “*limit=Inf*” value, each pass will continue until the end of media or file is reached.
- When the “*step=value*” option is used, limit controls the maximum offset stepped to.

Syntax:
```
limit=value    The number of bytes to transfer (data limit).
  or
limit=random   Random data limits between 10485760 and 2147483648 bytes.
```

Increment Limit “Incr_limit=“ Option

When using min/max limit options, this option specifies the increment value.

Syntax:
```
incr_limit=value    The data limit increment.
  or
incr_limit=random  Random data limit between min/max data limits.
```

```
  Example:
    $ dt of=dt.data bs=4k min_limit=512 max_limit=1m incr_limit=100k files=10
```

```
The above command will generate 10 files of starting with 512 bytes, then
incrementing by 100k for each file created. Add dispose=keep to see results.
```

Minimum Limit “min_limit=“ Option

This specifies the minimum data limit to be used in conjunction with max/incr limit options.

Syntax:
```
    min_limit=value    The minimum data limit number
```

Maximum Limit “max_limit=“ Option

This specifies the maximum data limit to be used in conjunction with min/incr limit options.

Syntax:
```
    max_limit=value    The maximum data limit.
```

Capacity “capacity=“ Option

This option allows the user to set the disk capacity (for raw disks), to a particular capacity, rather than using the capacity returned by the host OS or using *dt*’s automatic calculation logic, used during random and reverse I/O or with multiple slices options.

Syntax:
```
      capacity=value   Set the device capacity in bytes.
   or capacity=max     Set maximum capacity from disk driver.
```

Capacity Percentage “capacityp=” Option

This option is used to specify a percentage of the disk capacity to be used, rather than the full capacity. This is useful testing thinly provisioned storage, when there are insufficient back end disks.

Syntax:
```
      capacityp=value      Set capacity by percentage (range: 0-100).
```

Log File “log=“ Option

This option redirects dt’s output to a disk file, both standard error and standard output.

Syntax:
```
      log[tu]=filename The log file name to write.
                       t=truncate, u=unique (w/tpid)
```

Log File Format Control Strings

When specifying the log file, dt recognizes a number of format control strings.

Please see the Format Control section for a complete list of valid strings.

Example: log=dt_%host_%user_%iodir_%iotype-%pid.log

Log Prefix "logprefix=" Option

The per line logging prefix. This overrides *dt*'s default and allows user defined prefix strings.

```
Syntax:
  logprefix=string      The per line logging prefix.

Example:
  logprefix='%nos %et %prog (j:%job t:%thread): '
```

Error Log "error_log=" Option

The error log file. This file is written to whenever errors occur. Generally this file should be deleted, as necessary, prior to each set of tests. Since multiple *dt* processes may be utilized, this file is **not** deleted automatically, unless enable=deleteerrorlog option is specified.

```
Syntax:
  error_log=filename    The error log file name. (alias: elog=)
```

Master Log "master_log=" Option

For those who prefer to have all job and thread output written to a single file, the master log option is used. Please know that when multiple jobs/threads are writing, a single lock is used.

```
Syntax:
  master_log=filename   The master log file name. (alias: mlog=)
```

Reread File "reread_file" Option

This option is used to write command lines required to reread data *after* errors occur.

So how can this be used?
When controller outage is being verified, such as turning off controller power, executing *dt* with this reread file, will verify all data previously written up to the point of failure.

```
Syntax:
  reread_file=filename  The reread file name.
```

Common Open Flags "*flags*=" Option

Output Open Flags "*oflags*=" Option

These options are used to specify various POSIX compliant open flags, and system specific flags, to test the affect of these open modes.

Special Notes:

- .Each operating system has different flags, which can be queried by reviewing the *dt* help text ("*dt help*").

```
Syntax:
    flags=flags       Set open flags: {excl,sync,...}.
    oflags=flags      Set output flags: {append,trunc,...}.
```

Verify Flags "vflags=" Option

This option overrides the default block tag verify flags. The verify flags control which part of the block tag to verify during read operations. When the data being verified is *not* on the original file system or disk, you will need to specify vflags to disable so a compare error does not occur. For example, if you have copied a dt file to another location or server, and wish to re-verify the data, *vflags=~inode* will be required. Note: The '~' will clear flags rather than set flags.

```
Syntax:
  vflags=flags        Set/clear btag verify flags. {lba,offset,...}
```

Notes:

To see the valid verify flags, use "**`vflags=" option`**.

On Error Action "*onerr*=" Option

This option allows you to control the action taken by *dt* when an error occurs. By default, the action is *continue*, which allows all threads to run to completion. If the error action is set to *abort*, then *dt* stops all threads. If set the *pause*, then threads will be pause running.

```
Syntax:
    onerr=action     Set error action: {abort, continue, or pause}.
```

No Progress Time "*noprogt*=" Option

This option allows you to specify a time (in seconds) to report when I/O is not making progress. This option is used in conjunction with the "*alarm=*" option to periodically check for an report when I/O is taking too long. This is especially useful during controller failover type testing.

```
Syntax:
    noprogt=value    Set the no progress time (in seconds).
```

Example:

```
dt ... alarm=5s trigger="cmd:trigger" enable=noprog noprogt=120s
dt (16308): No progress made for 120 seconds!
dt (16308): Executing: trigger /var/tmp/dt.data-16308 noprog 512 131072 0 0 0
/var/tmp/dt.data-16308 noprog 512 131072 0 0 0
dt (16308): Trigger exited with status 2!
dt (16308): Sleeping forever...
  ...
```

In this example, an alarm() is set for every 5 seconds, and when the current I/O exceeds 120 seconds, a message is displayed and the trigger script is executed with "op = noprog". If the "*trigger=*" option were omitted, then only the warning message is displayed.

When the "*trigger=cmd*:..." option is utilized, the exit status controls the subsequent action to take: CONTINUE = 0, TERMINATE = 1, SLEEP = 2, or ABORT = 3

No Progress Time Trigger "*noprogtt=*" Option

This option allows you to specify a time (in seconds) when to initiate the no-progress time trigger script. Note: This option has no effect, unless the *noprogt=* option is enabled.

Syntax:
    noprogtt=value   Set the no progress time trigger (in seconds).

No Time "*notime=*" Option

This option allows you to disable timing of certain operations (system calls), when the no-progress options is enabled.

Special Notes:

- This option has no effect, unless the *noprogt=* option is enabled.
- `Valid optype's are: open close read write ioctl fsync msync aiowait`

Syntax:
    notime=optype   Disable timing of specified operation type.

Terminal Parity Setting "*parity=*" Option

This option specifies the terminal parity setting to use during testing.

```
  Syntax:
    parity=string      Set parity to: even, odd, or none.
 on QNX parity=string  Set parity to: even, odd, mark, space, or none.
```

Pass Limit "*passes=*" Option

This option controls the number of passes to perform for each test.

Special Notes:

- The default is to perform 1 pass.
- When using the "*of=*" option, each write/read combination is considered a single pass.
- When multiple passes are specified, a different data pattern is used for each pass, unless the user specified a data pattern or pattern file. [Please keep this in mind when using the "dispose=keep" option, since using this same file for a subsequent *dt* read verify pass, will report comparison errors... I've burnt myself this way. ☹]

Syntax:
    passes=value   The number of passes to perform.

Pass Command "*pass_cmd*=" Option

This option specifies a script to execute at the end of each pass. This allows test specific commands to be executed prior to starting the next pass.

```
Syntax:
   pass_cmd=string      The per pass command to execute.
```

Data Pattern "*pattern*=" Option

This option specifies a 32 bit hexadecimal data pattern to be used for the data pattern. *dt* has 12 built-in patterns, which it alternates through when running multiple passes. The default data patterns are:

0x39c39c39, 0x00ff00ff, 0x0f0f0f0f, 0xc6dec6de, 0x6db6db6d, 0x00000000,
0xffffffff, 0xaaaaaaaa, 0x33333333, 0x26673333, 0x66673326, 0x71c7c71c

You can also specify the special keyword "incr" to use an incrementing data pattern, or specify a character string (normally contained within single or double quotes).

```
Syntax:
    pattern=value    The 32 bit hex data pattern to use.
or  pattern=iot      Use DJ's IOT test pattern.
or  pattern=incr     Use an incrementing data pattern.
or  pattern=string   The string to use for the data pattern.
```

So, what is DJ's IOT test pattern? This pattern places the logical block address (lba) in the first word (4 bytes) of each block, with (lba+=0x01010101) being placed in all remaining words in the data block (512 bytes by default). In this way, the logical block is seeded throughout each word in the block. Note: The 4 byte lba needs increased to 8 bytes for larger capacity disks!

When specifying a pattern string via "pattern=string", the following special mapping occors:

```
Pattern String Mapping:
   \\ = Backslash   \a = Alert (bell)   \b = Backspace
   \f = Formfeed    \n = Newline        \r = Carriage Return
   \t = Tab         \v = Vertical Tab   \e or \E = Escape
   \ddd = Octal Value    \xdd or \Xdd = Hexadecimal Value
```

File Position "*position*=" Option

This option specifies a byte offset to seek to prior to starting each pass of each test.

```
Syntax:
   position=offset  Position to offset before testing.
```

Output File Position "*oposition*=" Option

This option specifies the output offset to position to during copy/very operations.

Syntax:
oposition=offset The output file position (copy/verify).

Prefix “*prefix*=“ Option

This option allows the user to define a free format prefix string which is written at the beginning of each block. It is used to generate uniqueness useful when data corruption occur. Certain format control strings are interpreted as shown below.

Syntax:
prefix=string The data pattern prefix string.

The prefix format controls permitted are:

```
Prefix Format Control:
        %d = The device name.           %D = The real device name.
        %h = The host name.             %H = The full host name.
        %p = The process ID.            %P = The parent PID.
        %u = The user name.

  Example: prefix="%u@%h (pid %p)"
```

Multiple Processes “*procs*=“ Option

This option specifies the number of processes to initiate performing the same test. This option allows an easy method for initiating multiple I/O requests to a single device or file system.

Special Notes:

- The per process limit on Tru64 UNIX is 64, and can be queried by using the “*sysconfig -q proc*” command.
- Spawning many processes can render your system useless, well at least very slow, and consumes large amounts of swap space (make sure you have plenty!).
- The parent process simply monitors (waits for) all child prcoesses.
- When writing to a file system, the process ID (PID) is appended to the file name specified with the “*of*=“ option to create unique file names. If no pattern is specified, each process is started with a unique data pattern. Subsequent passes cycle through the 12 internal data patterns. Use “*disable=unique*” to avoid this behaviour.
- The spawn() facility, used to execute on a different node, is not implemented on the QNX Operating System at this time.

Syntax:
procs=value The number of processes to create.

Set Queue Depth “*qdepth*=“ Option

This option is currently only implemented on HP-UX. It allow you to set the queue depth of the device under test, overriding its’ default. Note: The settings is sticky (retained).

Syntax:

qdepth=value Set the queue depth to specified value.

Random I/O Offset Alignment "*ralign*=" Option
This option is used when performing random I/O, to align each random block offset to a particular alignment, for example 32K.

Syntax:
ralign=value The random I/O offset alignment.

Random I/O Data Limit "*rlimit*=" Option
This option is used with random I/O to specify the number of bytes to limit random I/O between (starting from block 0 to this range). This option is independent of the data limit option.

Syntax:
rlimit=value The random I/O data byte limit.

Random Seed Value "*rseed*=" Option
This options sets the seed to initialize the random number generator with, when doing random I/O. When selecting random I/O, the total statistics displays the random seed used during that test. This option can be used to repeat the random I/O sequence of a test.

Syntax:
rseed=value The random seed to initialize with.

Record Limit "*records*=" Option
This option controls the number of records to process for each write and/or read pass of each test. The "*count*=" option is an alias for this option (supported for *dd* compatibility).

Special Notes:
- You must specify either a data limit, record limit, or files limit to initiate a test, unless the device type is "*disk*", in which case *dt* will automatically determine the disk capacity.
- When specifying a runtime via the "*runtime*=" option, the record limit controls how many records process for each pass (write and/or read pass).
- If you specify a infinite "*records=Inf*" value, each pass will continue until the end of media or file is reached.

Syntax:
records=value The number of records to process.

Read Percentage "readp=" Option
This option specifies the percentage of reads to perform. The remaining operations are writes.

Syntax:
```
readp=value   Percentage of accesses that are reads. Range [0,100].
              'random' keyword makes the read/write percentage random.
```

Random Percentage “randp=“ Option
This option specifies the percentage of random I/O to perform. The remaining operations are sequential.

Syntax:
```
  randp=value   Percentage of accesses that are random. Range [0,100].
```
Sequential access = 0%, otherwise random percentage.

Read Random Percentage “rrandp=“ Option
During the read pass, this option specifies the percentage of random reads performed.

Syntax:

rreadp=value   Percentage of read accesses that are random. Range [0,100].

Notes:

Data comparisons are usually disabled when using this option, unless a previous write pass has been done to populate the data with a known pattern. Remember, *dt* does **not** keep track of previous data written, so false mis-compares will occur if this is not kept in mind.

Write Random Percentage “wrandp=“ Option
During the write pass, this option specifies the percentage of random writes performed.

Syntax:

wrandp=value   Percentage of write accesses that are random. Range [0,100].

Run Time “*runtime*=“ Option
This option controls how long the total test should run. When used in conjunction with a data limit or record limit, multiple passes will be performed until the runtime limit expires. A later section entitled “*Time Input Parameters*”, describes the shorthand notation for time values.

Syntax:

runtime=time   The number of seconds to execute.

Retry Delay “retry_delay=“ Option
This option controls the number of seconds to delay between reads performed *after* a data corruption. (see enable=retryDC option)

Syntax:
```
  retry_delay=value Delay before retrying operation. (Def: 5)
```

Script File “script=“ Option
This option specifies a script file to read *dt* commands from. The default file extension is “.dt”. This provides a simple method to create custom workloads that require more than one sequence. This is also useful for specifying a script to test multiple disks or file systems, in conjunction with *dt*’s multiple jobs with background (async) operations.

Syntax:
script=filename The script file name to execute.

Slice "*slice*=" Option

This option is used with random access devices. This option is used in conjunction with the "*slices*=value" option, which divides the media into slices (see below), then "*slice=value*" defines the slice to do testing to. Since *dt* does the calculations, this simplifies simultaneous testing from multiple hosts to shared storage (usually a multi-initiator test requrement).

Syntax:	
slice=value	The specific disk or file slice to test.

Example:
slices=3 slice=2

Assuming three hosts have access to the same disk, the above would be used for host 2.

Slices "*slices*=" Option

This option is used with random access devices. This option divides the media into slices. Each slice contains a different range of blocks to operate on in a separate process. If no pattern is specified, then each slice is started with a unique data pattern. Subsequent passes alternate through *dt*'s 12 internal patterns.

Syntax:	
slices=value	The number of disk slices to test.

Note: This option can be used in conjuntion with multiple processes and/or asynchronous I/O options to generate a heavy I/O load, great for stress testing!

Record Skip "*skip*=" Option

This option specifies the numer of records to skip prior to starting each write and/or read pass of each test. The skips are accomplished by reading records.

Syntax:	
skip=value	The number of records to skip past.

Record Seek "*seek*=" Option

This option specifies the number of records to seek past prior to starting each write and/or read test. The seeks are accomplished by lseek()'ing past records, which is much faster than skipping when using random access devices.

Syntax:	
seek=value	The number of records to seek past.

Data Step “*step*=“ Option

This option is used to specify non-sequential I/O requests to random access devices. Normally, *dt* does sequential read & writes, but this option specifies that step bytes to be seeked past after each request.

Special Notes:

- The “*limit=value*” option can be used to set the maximum offset.

Syntax:

```
  step=value      The number of bytes seeked after I/O.
```

Statistics Level “stats==“ Option

This option controls the statistics level. By default, full statistics are enabled.

Syntax:

```
  stats=level      The stats level: {brief, full, or none}
```

Please reference the pass statistics flag as well, to enable brief pass statistics.

Stop On File “stopon==“ Option

This option specifies a user created file to instruct *dt* to stop tests in a controlled way, thereby allowing threads to report their statistics. This is much preferred over simply killing *dt*.

Syntax:

```
  stopon=filename       Watch for file existence, then stop.
```

Sleep “*sleep=“ Options

These options are used in *dt* scripts to inject sleeps prior to executing further commands.

option specifies a user created file to instruct *dt* to stop tests in a controlled way, thereby allowing threads to report their statistics. This is much preferred over simply killing *dt*.

Syntax:

```
  sleep=time            The sleep time (in seconds).
  msleep=value          The msleep time (in milliseconds).
  usleep=value          The usleep time (in microseconds).
```

Show Block Tag “showbtag“ Option

This option is used to display block tag data, be it a disk or a file. The count and offset options can be used to control how many blocks to display.

Syntax:

```
  showbtags opts...     Show block tags and btag data.
Example:
  $ dt if=dtbtags.data showbtags offset=5k count=1
```

Show File System LBA “showfslba“ Option

Show File System Map “showfsmap“ Option

This option is used to display the physical LBAs that files map to. This option works on Linux and Windows. On Windows you must run as Administrator.

Syntax:

```
 showfslba              Show file system offset to physical LBA.
 showfsmap              Show file system map extent information.

File System Map Format:
  showfslba [bs=value] [count=value] [limit=value] [offset=value]
                       Show FS offset(s) mapped to physical LBA(s)
                       The default is to show LBA for specified offset.
  showfsmap [bs=value] [count=value] [limit=value] [offset=value]
                       Show the file system map extent information.
                       The default is to show the full extent map.
```

Show Time “showtime=“ Option

This option provides a single way to convert a hex time value to human readable.

Syntax:

```
   showtime=value        Show time value in ctime() format.
```

Example:

```
$ dt showtime=0x67842a08
dt.exe (j:0 t:0): The time is: 1736714760 seconds => Sun Jan 12 20:46:00 2025
```

Show Verify Flags “showvflags=“ Option

This option is used to display block verify flags.

Syntax:

```
 showvflags=value      Show block tag verify flags set.

   Block Tag Verify Flags: (prefix with ~ to clear flag)
       lba,offset,devid,inode,serial,hostname,signature,version
       pattern_type,flags,write_start,write_secs,write_usecs,
       pattern,generation,process_id,thread_number,device_size
       record_index,record_size,record_number,step_offset,
       opaque_data_type,opaque_data_size,crc32

       default Disk: lba,devid,serial + common
       default File: offset,inode + common flags
       common Flags: hostname,signature,write_start,generation,
                     prcoess_id,job_id,thread_number,crc32

       Example: verifyFlags= or vflags=~all,lba,crc32
```

Threads "threads=" Option

This option specifies the number of threads to execute in each job. By default, only one thread is created, but multiple threads is necessary to generate higher I/O load. When testing with a file system, multiple unique file names are generated. For testing disks, please see the *slices*= option.

Syntax:
```
threads=value          The number of threads to execute.
```

Trigger Type "*trigger*=" Option

This option specifies a trigger action to take whenever an error occurs and/or when the no-progress time has been exceeded (see "*enable=noprog*"). It's main purpose is for triggering an anlyzer and/or stopping I/O by some means (panic, etc) when trouble-shooting.

Syntax:
```
trigger={br, bdr, lr, seek, cdb:bytes, cmd:str, and/or triage}
                          The triggers to execute on errors.
```

Trigger Types:

br = Execute a bus reset.
bdr = Execute a bus device reset.
lr = Execute a LUN reset.
seek = Issue a seek to the failing lba.
cdb = Specify a custom CDB to send.
cmd:string = Execute command with these args:
  string dname op dsize offset position lba errno
triage = Perform triage SCSI commands. (Inquiry, Test Unit Ready)

When specifying the "cmd:" type, which invokes a program/script, the following arguments are passed on the command line:

Format: **cmd dname op dsize offset position lba errno noprogtime**

Where:

dname = The device/file name.
op = open/close/read/write/miscompare/noprog
dsize = The device block size.
offset = The current file offset.
position = The failing offset within block.
lba = The logical block address (relative for FS).
errno = The error number on syscall errors.
noprogtime = The no-progress time (in seconds).

Trigger Action "*trigger_action*=" Option

When triggers are executed, the exit status usually specifies the action to take thereafter. This option can be used to specify a particular action to take, based on the value specified.

Syntax:
```
  trigger_action=value  The trigger action (for noprogs).

The trigger actions supported are:
  0 = continue, 1 = terminate, 2 = sleep, 3 = abort
```

Trigger On "*trigger_on*=" Option

```
The option controls when triggers are executed. The default is "all".

  Syntax:
    trigger_on={all, errors, miscompare, or noprogs} (Default: all)
                          The trigger control (when to execute).
```

Terminal Speed "*speed*=" Option

This option specifies the terminal speed (baud rate) to setup prior to initiating the test. Although *dt* supports all valid baud rates, some speeds may not be supported by all serial line drivers, and in some cases, specifying higher speeds may result in hardware errors (e.g., silo overflow, framing error, and/or hardware/software overrun errors). The valid speeds accepted by *dt* are:

```
        0         50         75        110        134        150
      200        300        600       1200       1800       2400
     4800       9600      19200      38400      57600     115200
```

Although a baud rate of zero is accepted, this is done mainly for testing purposes (some systems use zero to hangup modems). The higher baud rates are only valid on systems which define the Bxxxxx speeds in termios.h.

Special Notes:
- The default speed is 9600 baud.

Syntax:
```
  speed=value    The tty speed (baud rate) to use.
```

Terminal Read Timeout "*timeout*=" Option

This option specifies the timeout to use, in 10ths of a second, when testing terminal line interfaces. This is the timeout used between each character after the first character is received, which may prevent tests from hanging when a character is garbled and lost.

Special Notes:
- The default terminal timeout is 3 seconds.
- The default timeout is automatically adjusted for slow baud rates.

Syntax:
```
  timeout=value   The tty read timeout in .10 seconds.
```

Terminal Read Minimum "*ttymin*=" Option
This option specifies the minmum number of characers to read, sets the VMIN tty attribute.

Special Notes:
- The tty VMIN field normally gets sets to the value of the block size (*bs=value*).
- Note that on some systems, the VMIN field is an *unsigned char*, so the maximum value is 255.
- On QNX, this field is an *unsigned short*, so a maximum of 65535 is valid.

Syntax:
```
ttymin=value     The tty read minimum count (sets vmin).
```

Multiple Volumes "*volumes*=" Option

Multi-Volume Records "*vrecords*=" Option
These options are used with removal media devices, to define how many volumes and records on the last volume to process (i.e., tapes, etc). By using these options, you do not have to *guess* at a data limit or record limit, to overflow onto subsequent volumes. These options automatically sets the "*enable=multi*" option.

Syntax:
```
volumes=value    The number of volumes to process.
vrecords=value   The record limit for the last volume.
```

Other Commands
The following options are self-explanatory, I believe. The *help* text is usually the most up to date, since whenever changes are made, the help text is also updated (more than this user guide). Several of the commands below are to use in scripts, or for converting values.

Syntax:
```
help                  Display this help text.
eval EXPR             Evaluate expression, show values.
system CMD            Execute a system command.
!CMD                  Same as above, short hand.
shell                 Startup a system shell.
usage                 Display the program usage.
version               Display the version information.
```

I/O Behavior ("behavior=) Option
In addition to the default *dt* I/O, several other I/O behaviors are available.

The *dtapp* behavior mimics what an application may perform. A list of disks/files can be specified with an extended block tag to track previous I/O, useful for troubleshooting. The primary list can also be verified with a mirror list, useful for verifying synchronous mirroring. A verification mode can also be used to verify asynchronous mirroring.

The *hammer* and *sio* I/O behaviors are tools which have been integrated into dt from standalone tools, provided by NetApp as open source. *Hammer* is an excellent file system test, while *sio* is a performance test.

```
    I/O Behaviors:
        iobehavior=type       Specify the I/O behavior. (alias: iob=)
          Where type is:
            dt                The dt I/O behavior (default).
            dtapp             The dtapp I/O behavior.
            hammer            The hammer I/O behavior.
            sio               The simple I/O (sio) behavior.

    For help on each I/O behavior use: "iobehavior=type help"
```

Force Corruption ("corrupt_*=) Options

These options were added to force false corruptions into either write or read operations. The purpose is twofold: 1) provide an easy way to verify trigger scripts, and 2) a way to verify the corruption analysis and extended error reporting is correct (mainly for myself, the author).

```
  Force Corruption Options:
    corrupt_index=value   The corruption index. (Default: random)
    corrupt_length=value  The corruption length. (Default: 4 bytes)
    corrupt_pattern=value The corruption pattern. (Default: 0xfeedface)
    corrupt_step=value    Corruption buffer step. (Default: 0 bytes)
    corrupt_reads=value   Corrupt at read records. (Default: 13)
    corrupt_writes=value  Corrupt at write records. (Default: 0)
```

Job Control Options

These options are used for job control, for use in scripts or by advanced automation. Each *dt* command line is started as a job with one or more threads. Jobs can run in the foreground or background. When jobs are run in the background (async option), when various commands are available to control jobs. The Scripts directory has several examples of using jobs.

```
    Job Start Options:
        istate={paused,running} (Default: running)
                              Initial state after thread created.
        tag=string            Specify job tag when starting tests.

    Job Control Options:
        jobs[:full][={jid|tag}] | [job=value] | [tag=string]
                              Show all jobs or specified job.
        cancelall             Cancel all jobs.
        cancel={jid|tag} | [job=value] | [tag=string]
                              Cancel the specified job ID.
        modify[={jid|tag}] | [job=value] | [tag=string] [modify_options]
                              Modify all jobs or specified job.
        pause[={jid|tag}] | [job=value] | [tag=string]
                              Pause all jobs or specified job.
        query[={jid|tag}] | [job=value] | [tag=string] [query_string]
                              Query all jobs or specified job.
        resume[={jid|tag}] | [job=value] | [tag=string]
                              Resume all jobs or specified job.
        stopall               Stop all jobs.
        stop={jid|tag} | [job=value] | [tag=string]
                              Stop the specified job.
        wait[={jid|tag}] | [job=value] | [tag=string]
                Wait for all jobs or specified job.
```

File Locking Options

These options were added to verify file system locking. Both *dt* and *hammer* I/O behaviors honor file locking options.

```
  File Locking Options:
    enable=lockfiles     Enables file locks (locks & unlocks)
    lockmode={mixed | full | partial}
                         Chance of full or partial file locks (default: mixed).
    unlockchance=[0-100]  Probability of keeping locks and skipping unlocking.
    Examples:
      unlockchance=100  100% chance of unlocking, ALL files unlocked. [default]
      unlockchance=50   50% chance of unlocking each file.
      unlockchance=0    0% chance of unlocking, NO files are unlocked.
```

Define Workload ("define") Option

Workload ("workload=") Option

These options allow workloads to be defined and/or specified or displayed. This version of *dt* has several predefined workloads, that can be displayed with the *workloads* option.

As of this writing, there are over 60 predefined workloads. The reason predefined workloads were created was to provide "known to work" workloads, and to make it easier to specify advanced workloads without the very long set of options required. In most cases, using a predefine workload is a great starting place, then additional options can be used to override or add options.

The *define* command is used to create your own *dt* workload. This command is usually used in the *dt* startup file (".datatestrc"). Thereafter you can refer to this workload by name.

```
   Workload Options:
    define workloadName options...
                       Define a workload with options.
    workloads [substr]
                       Display the valid workloads.
    workload=name       Select the specified workload.
```

File System Full ("fsfree_*") Options

These options are used to control file system full behavior. The default are usually adequate, but depending on the storage array being tested, the freeing of blocks may be done by a background thread which may require increasing these delays.

```
File System Full Options:
  fsfree_delay=value    FS free space sleep delay.    (Def: 3 secs)
  fsfree_retries=value  FS free space wait retries.   (Def: 10)

  Please consider adding the truncate flag or enable=deleteperpass,
  to free space between passes or with multiple threads to same FS.
```

Retry Error ("retry_*") Options

These options control retry options. The "DC" option stand for Data Corruption.

```
Retry Related Options:
  retry_error=value     The error code to retry.
  retry_delay=value     The retry delay.            (Def: 5 secs)
  retry_limit=value     The retry limit.            (Def: 60)
  retryDC_delay=value   The retry corruptions delay.  (Def: 5)
  retryDC_limit=value   The retry corruptions limit.  (Def: 1)

Error Strings Accepted:
  EIO (5), ENXIO (6), EBUSY (16), ENODEV (19), ENOSPC (28), ESTALE (116)
    OR
  retry_error='*' or -1 to retry all errors.
```

SCSI Specific Options

These option are specific to SCSI operations, used for disk testing and/or performing SCSI I/O directly, rather than through the host disk driver.

Note: The *spt* tool, is required for the unmap operations.

```
  SCSI Specific Options:
    idt=string            The Inquiry device type. (both, device, or serial)
    spt_path=string       Path to SCSI (spt) tool.
    spt_options=string    Additional spt options.
    readtype=string       The SCSI read type (read8, read10, read16).
    writetype=string      The SCSI write type (write8, write10, write16,
writev16).
    scsi_recovery_delay=value The SCSI recovery delay.  (Def: 2 secs)
    scsi_recovery_retries=value The SCSI recovery retries.(Def: 60)
    scsi_timeout=value    The SCSI timeout (in ms).     (Def: 0ms)
    unmap_freq=value      The SCSI unmap frequency.     (Def: 0)
    unmap=type            The SCSI unmap type.
      Valid types are: random, unmap, write_same, zerorod.
```

Enable "*enable*=" and Disable "*disable*=" Options

These options are used to either enable or disable program flags which either alter default test modes, test actions, or provide additional debugging information. You can specify a single flag or multiple flags each seperated by a comma (e.g., "*enable=aio,debug,dump*").

Syntax:

enable=flag	Enable one or more of the flags below.
disable=flag	Disable one or more of the flags below.

The flags which can be enabled or disabled are described below.

POSIX Asynchronous I/O "*aio*" Flag

This flag is used to control use of POSIX Asynchronous I/O during testing, rather than the synchronous I/O read() and write() system calls.

Special Notes:

- Beware, you may need to rebuild *dt* on new versions of Tru64 Unix due to POSIX changes and/or AIO library changes between major releases.
- Reference the "*aios=*" option, for more special notes.

Flag:

aio	POSIX Asynchronous I/O.(Default: disabled)

Asynchronous Job "async" Flag

This flag is used to create a job in the background where it runs asynchronously. These jobs will either run to completion or job control options can pause/resume or stop running jobs.

Syntax:

```
  async      Asynchronous job control.  (Default: disabled)
```

Reporting Close Errors "*cerror*" Flag

This flag controls where close errors are reported as an error or a failure. When disabled, close errors are reported as a warning. This flag is meant to be used as a workaround for device drivers which improperly return failures when closing the device. Many system utilities ignore close failures, but when testing terminals and tapes, the close status us *very* important. For example with tapes, the close reflects the status of writing filemarks (which also flush buffered data), and the rewind status.

Flag:

cerrors	Report close errors. (Default: enabled)

Block Tag "btags" Flag

Block Tags create additional information at the start of each data block. This is used for unique data specific to the block, as well as information to aid with troubleshooting corruptions.

Flag:

btags	Block tag control flag. (Default: disabled)

Data Compare "*compare*" Flag

This flag disables data verification during the read pass of tests. This flag should be disabled to read to end of file/media to obtain maximum capacity statistics, or to obtain maximum performance statistics (less overhead).

Flag:		
compare	Data comparison.	(Default: enabled)

Extra Block Tag Compare "x*compare*" Flag

This flag is used in conjunction with block tags to perform extra comparison.

Flag:		
xcompare	Extra btag prefix compare.	(Default: disabled)

Core Dump on Errors "*coredump*" Flag

This flag controls whether a core file is generated, via abort(), when *dt* is exiting with a failure status code. This is mainly used for program debug and is not of much interest to normal users. When testing multiple processes, via fork(), this is useful if your OS debugger does not support debugging child processes.

Flag:		
coredump	Core dump on errors.	(Default: disabled)

Delete Error Log File "deleteerrorlog" Flag

This flag controls deleting the error log file upon startup. When using multiple *dt* processes you may wish to disable this flag, but when using a single *dt* instance testing multiple disks/files, this default is appropriate.

Flag"		
deleteerrorlog	Delete error log file.	(Default: enabled)

Dump Block Tags "dump_btags" Flag

This flag controls dumping block tag data in addition to the formatted human readable block tag.

Flag:		
dump_btags	Dump block tags (btags).	(Default: disabled)

Dump Context "dump_context" Flag

When displaying corrupted data, this flag controls dumping a good block prior to a bad block, when a good block proceeds the bad block. This is supported for the IOT data pattern only.

Flag:		
dump_context	Dump good context block.	(Default: enabled)

Report Errors “errors” Flag

This flag controls the reporting of errors. Best I can tell, this is historic since all errors are *always* reported, except for expected errors.

Flag:

errors	Report errors flag.	(Default: disabled)

Report Extended Errors “xerrors” Flag

This flag controls extended error reporting, which is usually preferred.

Flag:

xerrors	Report extended errors.	(Default: enabled)

File Per Thread “fileperthread” Flag

This flag controls creating a file per thread to ensure that each thread has a unique file name. Internally, this flag is disabled when using multiple slices to a single file.

Flag:

fileperthread	File per thread.	(Default: enabled)

File File Once “fill_once” Flag

Always Fill Files “fill_always” Flag

When doing file testing, this flag controls whether to prefill a file before testing. Internally, *dt* will enable this flag automatically when doing file performance testing, so actual data is read rather than accessing sparse data skewing results.

Flag:

fill_always	Always fill files.	(Default: disabled)
fill_once	Fill the file once.	(Default: disabled)

File System Map Control Flag “fsmap” Flag

This flag controls whether file system mapping is performed. The file system mapping converts file system offset to the underlying physical block, desired when troubleshooting corruptions. This mapping is supported for Linux and Windows NTFS file systems.

Flag:

fsmap	File system map control.	(Default: enabled)

File System Trim “fsmap” Flag

On Windows this controls whether file system trim operations are performed. A file system trim is converted to a SCSI UNMAP operation, for example, which informs intelligent storage arrays to free storage blocks.

Flag:

fstrim	File system trim.	(Default: disabled)

File System Increment “fsincr” Flag

When testing multiple files, this flag controls increasing the size of each file by the block size.

Flag:

fsincr	File size incrementing.	(Default: disabled)

Log File Trailer “fsincr” Flag

This flag controls reporting the initial information, such as SCSI specific information, at the end of the log file.

Flag:

trailer	Log file trailer.	(Default: enabled)

Force Corruption “force-corruption” Flag

This flag is mainly used for testing trigger scripts and verifying corruption error reporting. You can create incorrect data or inject read corruptions, using this flag in conjunction with other corruption options.

Flag:

force-corruption	Force a FALSE corruption.	(Default: disabled)

Image Mode Copy “image” Flag

During copy operations, this flag ensures the source and destination capacity will be the same, thus ensuring an exact image copy is made. This flag is used when disks are different sizes.

Flag:

image	Image mode copy (disks).	(Default: disabled)

I/O Lock “iolock” Flag

This flag is used with multiple threads to coordinate I/O operations. When enabled, threads compete for an I/O to determine the next offset. Without this flag, *dt* will normally use slices which means threads create random I/O. With fixed media this I/O lock avoids excessive head movement with sequential I/O, provided high performance. Please note however, too many threads create lock contention. I have found roughly 8-10 threads give best performance.

Flag:

iolock	I/O lock control.	(Default: disabled)

Lock Files “lockfiles” Flag
This flag control file system lock operations. This works with local storage, of course, but is most beneficial for network file systems (IMO).

Flag:		
lockfiles	Lock files.	(Default: disabled)

Millisecond Delay “msecdelay” Flag
This flag enables millisecond delays with using various delay options. For random access devices (disks and files), microsecond delays are set by default. This flag overrides the default.

Flag:		
msecsdelay	Millisecond delays.	(Default: disabled)

Second Delays “secsdelay” Flag
This flag enables second delays when using the various delay options.

Flag:		
secsdelay	Second delays.	(Default: disabled)

Mount Lookup “mount_lookup” Flag
This flag controls mount device lookups on Unix systems. Mount lookups are used to acquire additional information for the file system under test. This information is displayed with the header and trailer flags. This flag can disable this lookup, if problems are encountered.

Flag:		
mount_lookup	Mount device lookup.	(Default: enabled)

Pipe Mode Control “pipes” Flag
This flag is used to enable pipe interaction. When enabled, *dt* will stay in an interactive mode, and emit a special prompt that scripts can utilize to know when to send additional commands. The Scripts/dt.ksh script provides and example of using pipes. Advanced automation using other scripting languages can also utilize pipe mode, along with job control to minimize the number of *dt* processes executed using CLI commands only. This method allows better scale up on hosts.

Flag:		
pipes	Pipe mode control flag.	(Default: disabled)

Poison Read Buffer “poison” Flag
This flag modifies the first 4 bytes of each block in a read buffer to a known pattern. This is useful to ensure that the read operation overwrites the previous data which can be misleading.

Flag:		
poison	Poison read buffer flag.	(Default: disabled)

Image Mode Copy "job_stats" Flag

This flag controls the job statistics. When multiple threads are specified, job statistics is an accumulation of all the thread statistics. When thread statistics are too verbose, job statistics will reduce the size of the log file. But please know that for debugging, thread statistics are generally very used (per those reviewing log file failures 😊)

Flag:

job_stats	The job statistics flag.	(Default: disabled)

Total Statistics "total_stats" Flag

After multiple passes, via passes= or runtime= options, *dt* emits total statistics. This flag can disable this reporting. But please know that total statistics are often useful for debugging.

Flag:

total_stats	The total statistics.	(Default: enabled)

Re-Read After Read-After-Write "reread" Flag

This flag controls an extra read pass after performing a read-after-write pass. This is important because data may be correct after an immediate read after the write, but later corrupted after some time elapses, since data written is cached and is often written to backend storage later.

Flag:

reread	Re-read after raw.	(Default: disabled)

Note: For highest data verification, please enable this flag with read-after-write pass.

Restart File System Full "resfsfull" Flag

This flag controls restarting a test after a file system full.

Note: While attempts are made to handle no space conditions when enabled, with multiple threads this action may not succeeed, but in most cases id does.

Flag:

resfsfull	Restart file system full.	(Default: enabled)

Retry Session Disconnects "retrydisc" Flag

On Windows, when testing network storage, session disconnects may occur, either plan or unplanned due to negative testing. Internally, *dt* recognizes a set of error code which may indicate a session disconnect, which then causes the operation to be retried rather than reporting an error.

Flag:

retrydisc	Retry session disconnects.	(Default: disabled)

Retry Warning “retrywarn” Flag

When retrying an error, *dt* will normally report this as an ERROR then retry the operation, as instructed by retry options. But if negative testing is causing the error, then this flag will report the message as a warning and avoid counting the error.

Flag:

retrywarn	Retry logged as warning.	(Default: disabled)

Save Corrupted Data “savecorrupted” Flag

This flag controls saving corrupted data to a separate file. Both the expected and corrupted data is saved, which may be useful for later analysis. Generally this is desirable, but can be disabled.

Flag:

savecorrupted	Save corrupted data.	(Default: enabled)

Script Verify “scriptverify” Flag

This flag controls displaying the commands in a dt script file, which is handy when failures occur. This is analogous to the Unix shell “-x” show execution flag.

Flag:

scriptverify	Script verify display.	(Default: disabled)

Hangup Signal Control “sighup” Flag

This flag controls how *dt* handles a hung up signal (SIGHUP). For example when starting a set of *dt* commands in the background, then logging off, a SIGHUP is generated. Ignoring this signal allows *dt* processes to continue running, rather than exiting.

Flag:

sighup	Hangup signal control.	(Default: enabled)

Sparse File Attribute “sparse” Flag

Pre-Allocate Without Sparse “prealloc” Flag

This flag controls setting the Windows sparse flag when random or reverse I/O is detected.

Why is this important, you may ask?
On Windows, if a write is done to the end of the file, a pre-allocation operation occurs. When no-prog options are enabled, this pre-allocation can trigger *false* no-prog messages. Setting the sparse attribute prior to opening the file, simulates a sparse file similar to Unix file systems.

Note: Internally, *dt* sets spares automatically when random or reverse I/O is detected.
The pre-allocate without sparse, will write file data to cause pre-allocation.

Flag:

sparse	Sparse file attribute.	(Default: enabled)
prealloc	Preallocate w/o sparse.	(Default: enabled)

SCSI Operation "scsi*" Flags
These flags are used to control various SCSI operations. When SCSI I/O is desired, to bypass the host disk driver, the "scsi_io" flag must be enabled. Other flag control gathering and reporting various SCSI information, controlling debug and error reporting, and/or performing SCSI UNMAP commands.

Flags:
```
  scsi              SCSI operations.           (Default: enabled)
  scsi_info         SCSI information.          (Default: enabled)
  scsi_io           SCSI I/O operations.       (Default: disabled)
  sdebug            SCSI debug output.         (Default: disabled)
  scsi_errors       SCSI error logging.        (Default: disabled)
  scsi_recovery     SCSI recovery control.     (Default: enabled)
  unmap             SCSI unmap per pass.       (Default: disabled)
  get_lba_status    SCSI Get LBA Status.       (Default: disabled)
  fua               SCSI Force unit access.    (Default: disabled)
  dpo               SCSI Disable page out.     (Default: disabled)
```

Please Note:
SCSI UNMAP and Get LBA Status commands required the *spt* tool, also open source.

Stop Immediate w/Stop File "stopimmed" Flag
This flag controls whether *dt* will exit immediately when the external stop on file is detected, or waiting until a full write/read pass completes. When testing large media, and using the standard write then read/verify passes, it's possible the read/verify pass never occurs with short runtimes. That said, on the other hand, waiting for a full write/read pass may take a long time, especially with a heavily loaded host or storage array with many hosts doing I/O.

Flag:

stopimmed	Stop immediate w/stop file.	(Default: enabled)

Terminate on Signals "terminate_on_signals" Flag
Normally *dt* catches a set of signals and on the first signal set the stop state for each thread, in attempts to have threads report their statistics and exit normally. A second signal will then cause *dt* to exit immediately without further waiting. This flag will enable the latter behavior.

Flag:

terminate_on_signals	Terminate on signals.	(Default: disabled)

Trigger Control "trig*" Flags
These flags control various trigger actions. Used for specialized test cases.

Flags:

trigargs	Trigger cmd arguments.	(Default: enabled)
trigdefaults	Automatic trigger defaults.	(Default: enabled)
trigdelay	Delay mismatch triggers.	(Default: enabled)

UUID Dashes "uuid_dashes" Flag
The Unique User ID format control strings normally generate a long string like this:

ef0d7ef1-898b-4f45-b476-710e66a7cc44

This option will disable the "-" so a long hex numeric value is generated instead:
ef0d7ef1898b4f45b476710e66a7cc44

Flag:

uuid_dashes	Dashes in UUID strings.	(Default: enabled)

Diagnostic Logging "*diag*" Flag

This option is only valid on Tru64 Unix. When enabled, error messages get logged to the binary error logger. This is useful to correlate device error entries with test failures. Please note, the logging only occurs when running as superuser (API restriction, not mine!).

Flag:

diag	Log diagnostic msgs.	(Default: disabled)

Debug Output "*debug*" Flag

Verbose Debug Output "*Debug*" Flag

Other Debug Output "**debug*" Flags

These flags are used to enable debug output for different operations. Multiple flags can be specified via a comma separated list.

```
Flags:
  debug            Debug output.              (Default: disabled)
  Debug            Verbose debug output.      (Default: disabled)
  btag_debug       Block tag (btag) debug.    (Default: disabled)
  edebug           End of file debug.         (Default: disabled)
  fdebug           File operations debug.     (Default: disabled)
  jdebug           Job control debug.         (Default: disabled)
  ldebug           File locking debug.        (Default: disabled)
  mdebug           Memory related debug.      (Default: disabled)
  mntdebug         Mount device lookup debug. (Default: disabled)
  pdebug           Process related debug.     (Default: disabled)
  rdebug           Random debug output.       (Default: disabled)
  tdebug           Thread debug output.       (Default: disabled)
  timerdebug       Timer debug output.        (Default: disabled)
```

Delete Per Pass "*deleteperpass*" Flag

The deleteperpass option deletes test files between multiple passes. This is especially handy when testing file system full/quota exceeded conditions, to start clean on each pass. Otherwise, even with file truncation, subsequent passes may encounter early file system full conditions, esp. with random I/O, which may not be very interesting.

Flag:

deleteperpass	Delete files per pass.	(Default: disabled)

Dump Data Buffer “*dump*” Flag

This flag controls dumping of the data buffer during data comparision failures. If a pattern file is being used, then the pattern buffer is also dumped for easy comparision purposes. To prevent too many bytes from being dumped, esp. when using large block sizes, dumping is limited to 512 bytes of data (was 64, recently increased).

Special Notes:

- When failures occur within the first 64 bytes of the buffer, dumping starts at the beginning of the buffer.
- When the failure occurs at some offset within the data buffer, then dumping starts at (data limit/2) bytes prior to the failing byte to provide context.
- The start of the failing data is marked by an asterisk '*'.
- You can use the *dlimit=* option to override the default dump limit.
- Buffer addresses are displayed for detection of memory boundary problems.

Flag:

dump	Dump data buffer.	(Default: enabled)

Tape EEI Reporting “*eei*” Flag

This option controls the reporting of Extended Error Information (EEI) on **Tru64 UNIX** systems, for tape devices when errors occur. The standard tape information available from *mt* is reported, along with the EEI status, CAM status, and SCSI request sense data. This is excellent information to help diagnose tape failures. (thank-you John Meneghini!)

Flag:

eei	Tape EEI reporting.	(Default: enabled)

Flush Terminal I/O Queues “*flush*” Flag

This flag controls whether the terminal I/O queues get flushed before each test begins. This must be done to ensure no residual characters are left in the queues from a prior test, or else data verification errors will be reported. Residual characters may also be left from a previous XOFF’ed terminal state (output was suspended).

Flag:

flush	Flush tty I/O queues.	(Default: enabled)

History Dumping “*hdump*” Flag

This flag controls dumping the history entries at the end of a test. Normally dt only dumps the history during errors, but this option when enabled, dumps the history when exiting. This is useful if you are timing I/O’s, or wish to see the LBA’s I/O went to, etc.

Flag:

hdump	History dump.	(Default: disabled)

History Timing “*htiming*” Flag

This flag controls the timing of history entries. Please be aware, that enabling timing of each I/O will impact your overall test performance, as an extra system call is used to obtain system time.

Flag:

htiming	History timing.	(Default: disabled)

Log File Header “*header*” Flag

When a log file is specified, *dt* automatically writes the command line and *dt* version information at the beginning of the log file. This option allows you to control whether this header should be written.

Flag:

header	Log file header.	(Default: enabled)

Loop On Error “*looponerror*” Flag

This flag controls lopping on data corruption rereads. This can be helpful in capturing the failing read request on an analyzer.

Special Notes:

- Also see “retry_delay=value” and retryDC flag control.

Flag:

looponerror	Loop on error.	(Default: disabled)

Logical Block Data Mode “*lbdata*” Flag

This option enables a feature called logical block data mode. This feature allows reading/writing of a 4-byte (32-bit) logical block address at the beginning of each data block tested. The block number is stored using SCSI byte ordering (big-endian), which matches what the SCSI Write Same w/lbdata option uses, so *dt* can verify this pattern, generated by *scu*’s “*write same*” command.

Special Notes:

- The starting logical block address defaults to 0, unless overridden with the “*lba=*“ option.
- The logical block size defaults to 512 bytes, unless overridden with the “*lbs=*“ option.
- The logical block address is always inserted started at the beginning of each data block.
- Enabling this feature will degrade performance statistics (slightly).

Enable Loopback Mode “*loopback*” Flag

This flag specifies that either the input or output file should be used in a loopback mode. In loopback mode, *dt* forks(), and makes the child process the reader, while the parent process becomes the writer. In previous versions of *dt*, you had to specify both the same input and output file to enable loopback mode. When specifying this flag, *dt* automatically duplicates the input or output device, which is a little cleaner than the old method (which still works).

Warning: Terminal support is not known to work with this version of *dt*!

Some people may argue that *dt* should automatically enable loopback mode when a single terminal or FIFO device is detected. The rationale behind not doing this is described below:

1. You may wish to have another process as reader and/or writer (which also includes another program, not necessarily *dt*).
2. You may wish to perform device loopback between two systems (e.g., to verify the terminal drivers of two operating systems are compatible).
3. A goal of *dt* is *not* to force (hardcode) actions or options to make the program more flexible. A minimum of validity checking is done to avoid being too restrictive, although hooks exists to do this.

Special Notes:

- The read verify flag is automatically disabled.
- This mode is most useful with terminal devices and/or FIFO's (named pipes).

Microsecond Delays "*microdelay*" Flag

This flag tells *dt* that delay values, i.e. "*sdelay*=" and others, should be executed using microsecond intervals, rather the second intervals.

Flag:

microdelay	Microsecond delays.	(Default: disabled)

Memory Mapped I/O "*mmap*" Flag

This flag controls whether the memory mapped API is used for testing. This test mode is currently supported on SUN/OS, Tru64 UNIX, and Linux operating systems.

Special Notes:

- The block size specified "bs=" *must* be a multiple of the system dependent page size (normally 4k or 8k).
- An msync() is done after writing and prior to closing to force modified pages to permanent storage. It may be useful to add an option to inhibit this action at some point, but my testing was specifically to time mmap performance. Obviously, invalidating the memory mapped pages, kind of defeats the purpose of using memory mapped files in the first place.
- Specifying multiple passes when doing a read verify test, gives you a good indication of the system paging utilization on successive passes.
- Memory mapping large data files (many megabytes) may exhaust certain system resources. On an early version of SUN/OS V4.0?, I could hang my system by gobbling up all of physical memory and forcing paging (this was certainly a bug which has probably been corrected since then).

Flag:

mmap	Memory mapped I/O.	(Default: disabled)

Test Modem Lines "*modem*" Flag

This flag controls the testing of terminal modem lines. Normally, *dt* disables modem control, via setting CLOCAL, to prevent tests from hanging. When this flag is enabled, *dt* enables modem control, via clearing CLOCAL, and then monitoring the modem signals looking for either carrier detect (CD) or dataset ready (DSR) before allowing the test to start.

Warning: Terminal support is not known to work with this version of *dt*!

Special Notes:

- The program does not contain modem signal monitoring functions for the all operating systems. The functions in *dt* are specific to Tru64 UNIX and ULTRIX systems, but these can be used as templates for other operating systems.

Flag:

modem	Test modem tty lines. (Default: disabled)

Multiple Volumes "*multi*" Flag

This flag controls whether multiple volumes are used during testing. When this flag is enabled, if the data limit or record count specified does not fit on the current loaded media, the user is prompted to insert the next media to continue testing. Although this is used mostly with tape devices, it can be used with any removeable media.

Flag:

multi	Multiple volumes. (Default: disabled)

No I/O Progress "*noprog*" Flag

This flag controls whether *dt* will check for slow or no I/O progress during testing.

Special Notes:

- Enabling this flag will do nothing by itself. The "*alarm=*" option specifies the frequency of how often *dt* checks for no progress.
- The "*noprogt=secs*" option specified the no I/O progress time.
- If "*noprogt=*" is omitted, it defaults to the "*alarm=*" time value.
- The *noprog* flag is implicitly enabled by the "*noprogt=value*" option.

Flag:

noprog	No progress check. (Default: disabled)

Prefill "*prefill*" Flag

This flag controls the buffer prefill normally performed prior to reads. Normally, dt prefills the buffer with the inverted data pattern (1st four bytes). This, of course, ensures the data is overwritten with data read, but also imposes overhead not always desirable.

Special Notes:

- When IOT pattern is used, this flag is automatically enabled, since IOT blocks are unique.

Flag:

prefill	Prefill read buffer.	(Default: enabled)

Control Per Pass Statistics "*pstats*" Flag
This flag controls whether the per pass statistics are displayed. If this flag is disabled, a single summary line is still displayed per pass and the total statistics are still displayed in the full format.

Flag:

pstats	Per pass statistics.	(Default: enabled)

Read After Write "*raw*" Flag
This flag controls whether a read-after-write will be performed. Sorry, *raw* does **not** mean character device interface. Normally *dt* performs a write pass, followed by a read pass. When this flag is enabled the read/verify is done immediately after the write.

Flag:

raw	Read after write.	(Default: disabled)

Tape Reset Handling "*resets*" Flag
This option is used during SCSI bus and device reset testing, to reposition the tape position (tapes rewind on resets), and to continue testing. This option is only enabled for Tru64 UNIX systems (currently), since this option requires reset detection from EEI status, and tape position information from the CAM tape driver (although *dt* also maintains the tape position as a sanity check against the drivers' data).

Flag:

resets	Tape reset handling.	(Default: disabled)

Retry Data Corruptions "*retryDC"* Flag
This flag controls whether a data corruption retry is performed. A second read is done to re-read the data, with direct I/O for file systems, and the data is compared against the previous read data, and the expected data. If the reread data matches the expected data, then dt assumes a "read failure" occurred, otherwise if the reread data matches the previous read, dt assumes a "write failure" (the data was written incorrectly).

Flag:

retryDC	Retry data corruptions.(Default: enabled)	

Control Program Statistics "*stats*" Flag
This flag controls whether any statistics get displayed (both pass and total statistics). Disabling this flag also disabled the pass statistics described above.

Flag:

stats	Display statistics.	(Default: enabled)

Table(sysinfo) timing “*table*” Flag

On Tru64 UNIX systems, this option enables additional timing information which gets reported as part of the statistics display. (thanks to Jeff Detjen for adding this support!)

Flag:

table	Table(sysinfo) timing. (Default: disabled)

System Log “syslog” Flag

This flag controls logging startup/finish and errors being logged to the system logger. This can be helpful when correlating dt’s errors with system (driver/file system) error messages.

Flag:

syslog	Log errors to syslog. (Default: disabled)

Timestamp Blocks “*timestamp*” Flag

This flag controls whether blocks are timestamped when written. The timestamp is skipped during data comparisions, but *is* displayed if any remaining data is incorrect.

Special Notes:

- When IOT or lbdata patterns are used, the block number is overwritten by the timestamp.
- This flag is a stop-gap, until block tagging (w/more information) is implemented.

Flag:

timestamp	Timestamp each block. (Default: disabled)

Unique Pattern “*unqiue*” Flag

This flag controls whether multiple process, get a unqiue data pattern. This affects processes started with the “*slices=*“ or the “*procs=*“ options. This only affects the *procs=* option when writing to a regular file.

Flag:

unique	Unique pattern.	(Default: enabled)

Verbose Output “*verbose*” Flag

This flag controls certain informational program messages such as reading and writing partial records. If you find these messages undesirable, then they can be turned off by disabling this flag. *But beware, partial reads or writes of disk records if not at EOF is usually a problem!*

Flag:

verbose	Verbose output.	(Default: enabled)

Verify Data "verify" Flag

This flag controls whether the read verify pass is performed automatically after the write pass. Ordinarily, when specifying an output device via the "*of*=" option, a read verify pass is done to read and perform a data comparision. If you only wish to write the data, and omit the data verification read pass, then di able this flag.

Flag:

verify	Verify data written.	(Default: enabled)

Special Notes:

- If you don't plan to ever read the data being written, perhaps for performance reasons, specifying "*disable=compare*" prevents the data buffer from being initialized with a data pattern.
- This verify option has no affect when reading a device. You must disable data comparsions via "*disable=compare*".

Program Delays

dt allows you to specify various delays to use at certain points of the test. These delays are useful to slow down I/O requests or to prevent race conditions when testing terminals devices with multiple processes, or are useful for low level driver debugging. All delay values are in seconds, unless you specify "*enable=microdelay*", to enable micro-second delays.

Close File "*close_delay*=" Delay

This delay, when enabled, is performed prior to closing a file descriptor.

Delay

close_delay=value	Delay before closing the file.	(Def: 0)

Delete File "*delete_delay*=" Delay

This delay controls how long to wait prior to deleting files between pass or at end of test.

Delay:

delete_delay=value	Delay after deleting files.	(Default: 0 secs)

End of Test "*end_delay*=" Delay

This delay, when enabled, is used to delay after closing a device, but prior to re-opening the device between multiple passes.

Delay:

end_delay=value	Delay between multiple passes.	(Def: 0)

Open "open_delay=" Delay

This is a delay applied *before* opening a file.

Delay:

open_delay=value	Delay before opening the file.	(Default: 0)

Read Record "*rread_delay*=" Delay
This delay, when enabled, is used prior to issuing each read request (both synchronous read()'s and asynchronous aio_read()'s).

```
  Delay:
    read_delay=value    Delay before reading each record. (Def: 0)
```

Start Test "*start_delay*=" Delay
This delay, when enabled, is used prior to starting the test.

When testing terminal devices, when not in self-loopback mode, the writing process (the parent) automatically delays 1 second, to allow the reading process (the child) to startup and setup its' terminal characteristics. If this delay did not occur prior to the first write, the reader may not have its' terminal characteristics (flow, parity, & speed) setup yet, and may inadvertently flush the writer's data or receive garbled data.

Warning: Terminal testing is not known to work with this version of *dt*.

```
  Delay:
    start_delay=value    Delay before starting the test.   (Def: 0)
```

Terminate Thread "*term_delay*=" Delay
```
This delay controls how long to delay before a thread exits.

  Delay:
    term_delay=value  Delay before thread terminates.  (Default: 0 secs)
```

Terminate Wait "*term_wait*=" Time
```
This time controls long to wait for a thread to terminate gracefully when
instructed to stop. If the thread is hung, when this time is exceeded, steps
are taken to cancel the thread.

  Delay:
    term_wait=time   Thread termination wait time.     (Default: 180 secs)

Note:
  The default value is historically set for SAN storage per various storage
arrays. For other storage this may need increased, for example cloud storage,
or disabled by setting this to zero.
```

Verify "verify_*delay*=" Delay
```
The delay controls the amount of time to wait prior to verifying data.

  Delay:
    verify_delay=value  Delay before verifying data.      (Default: 0)
```

Write Record "*write_delay*=" Delay
This delay, when enabled, is used prior to issuing each write request (both synchronous write()'s and asynchronous aio_write()'s).

Delay:
  write_delay=value Delay before writing each record. (Def: 0)

Numeric Input Parameters
For any options accepting numeric input, the string entered may contain any combination of the following characters:

```
Special Characters:
    w = words (4 bytes)            q = quadwords (8 bytes)
    b = blocks (512 bytes)         k = kilobytes (1024 bytes)
    m = megabytes (1048576 bytes)  p = page size (8192 bytes)
    g = gigabytes (1073741824 bytes)
    t = terabytes (1099511627776 bytes)
    d = device size (set via dsize=value option)
    inf or INF = infinity (18446744073709551615 bytes)

Arithmetic Characters:
    + = addition                   - = subtraction
    * or x = multiplication        / = division
    % = remainder

Bitwise Characters:
    ~ = complement of value       >> = shift bits right
   << = shift bits left            & = bitwise 'and' operation
    | = bitwise 'or' operation     ^ = bitwise exclusive 'or'
```

The default base for numeric input is decimal, but you can override this default by specifying 0x or 0X for hexadecimal conversions, or a leading zero ‘0’ for octal conversions.

Time Input Parameters
When specifying the run time “runtime=“ option, the time string entered may contain any combination of the following characters:

Time Input:
  d = days (86400 seconds), h = hours (3600 seconds)
  m = minutes (60 seconds), s = seconds (the default)

Implicit addition is performed on strings of the form '1d5h10m30s', thus 1 day, 5 hours, 10 minutes, 30 seconds in this example.

Future Enhancements
Now that I’m retired, future enhancements will need to be requested, and depending on the time required to implement feature, I may be asking for a short-term contract, with NDA as required.

Initially *dt* was written to be a generic test tool, designed to test any device, and although that was (mostly) accomplished, device specific tests needed to be and were developed, based on the device type detected or specified by the "*dtype=*" option if not determined automatically.

Final Comments

I'm happy to report that *dt* is getting wide spread use all over the world! Storage groups, terminal/lat groups, Q/A, developers, and other peripheral qualification groups are using *dt* as part of their testing. I guess maybe this will be my (computer) legacy? ☺

Anyways, I hope you find *dt* as useful as I have. This is usually one of the first tools I port to a new operating system, since it's an excellent diagnostic and performance tool (it gives me a warm and fuzzy feeling ☺).

Please send me mail on any problems or suggestions you may have, and I'll try to help you out. The future development of *dt* depends a lot on user interest. Many of *dt*'s features have come about from user requests.

IF YOU LIKE MY WORK,
YOU CAN DO
ONE OF TWO THINGS:

THROW MONEY OR APPLAUD*
(*OR HIRE ME AND ALLOW ME TO WORK REMOTELY FROM MESQUITE, NV?*)

***OK, I'VE HEARD ENOUGH APPLAUSE!** 😊

Appendix A *dt* Help Text

The following help text is contained within the *dt* program. The help shown is for Linux, so some text varies for Windows. Please review files in the Documentation directory for additional documents.

```
% dt help
Usage: dt options...

    Where options are:
        if=filename           The input file to read.
        of=filename           The output file to write.
        sdsf=filename         The SCSI device special file.
        tdsf=filename         The SCSI trigger device file.
        pf=filename           The data pattern file to use.
        dir=dirpath           The directory path for files.
        dirp=string           The directory prefix for subdirs.
        filepostfix=str       The file postfix. (D: j%jobt%thread)
        sdirs=value           The number of subdirectories.
        depth=value           The subdirectory depth.
        bs=value              The block size to read/write.
    or  bs=random             Random sizes between 512 and 1048576 bytes.
        ibs=value             The read block size. (overrides bs=)
        obs=value             The write block size. (overrides bs=)
        job_log=filename      The job log file name. (alias: jlog=)
        logdir=filename       The log directory name.
        log[atu]=filename     The thread log file name to write.
                              a=append, t=truncate, u=unique (w/tid)
        logprefix=string      The per line logging prefix.
        error_log=filename    The error log file name. (alias: elog=)
        master_log=filename   The master log file name. (alias: mlog=)
        reread_file=filename  The reread file name.
        aios=value            Set number of AIO's to queue.
        alarm=time            The keepalive alarm time.
        keepalive=string      The keepalive message string.
        keepalivet=time       The keepalive message frequency.
        pkeepalive=str        The pass keepalive message string.
        tkeepalive=str        The totals keepalive message string.
        align=offset          Set offset within page aligned buffer.
    or  align=rotate          Rotate data address through sizeof(ptr).
        capacity=value        Set the device capacity in bytes.
    or  capacity=max          Set maximum capacity from disk driver.
        capacityp=value       Set capacity by percentage (range: 0-100).
        bufmodes={buffered,unbuffered,cachereads,cachewrites}
                              Set one or more buffering modes. (Default: none)
        boff=string           Set the buffer offsets to: dec or hex. (Default: hex)
        dfmt=string           Set the data format to: byte or word. (Default: word)
        dispose=mode          Set file dispose to: {delete, keep, or keeponerror}.
        dlimit=value          Set the dump data buffer limit.
        dtype=string          Set the device type being tested.
        idtype=string         Set input device type being tested.
        odtype=string         Set output device type being tested.
        dsize=value           Set the device block (sector) size.
        errors=value          The number of errors to tolerate.
        files=value           Set number of disk/tape files to process.
        maxfiles=value         The maximum files for all directories.
        ffreq=value            The frequency (in records) to flush buffers.
        fstrim_freq=value      The file system trim frequency (in files).
        fill_pattern=value     The write fill pattern (32 bit hex).
        prefill_pattern=value  The read prefill pattern (32 bit hex).
        flow=type              Set flow to: none, cts_rts, or xon_xoff.
        incr=value             Set number of record bytes to increment.
    or  incr=variable          Enables variable I/O request sizes.
        iops=value             Set I/O per second (this is per thread).
        iodir=direction        Set I/O direction to: {forward, reverse, or vary}.
        iomode=mode            Set I/O mode to: {copy, mirror, test, or verify}.
        iotype=type            Set I/O type to: {random, sequential, or vary}.
        iotpass=value          Set the IOT pattern for specified pass.
        iotseed=value          Set the IOT pattern block seed value.
        iotune=filename        Set I/O tune delay parameters via file.
        history=value          Set the number of history request entries.
        history_bufs=value     Set the history data buffers (per request).(or hbufs)
        history_bsize=value    Set the history block data size increment. (or hbsize)
        history_data=value     Set the history data size (bytes to save). (or hdsize)
        min=value              Set the minumum record size to transfer.
        max=value              Set the maximum record size to transfer.
        lba=value              Set starting block used w/lbdata option.
        lbs=value              Set logical block size for lbdata option.
        limit=value            The number of bytes to transfer (data limit).
    or  limit=random           Random data limits between 10485760 and 2147483648
bytes.
        incr_limit=value       Set the data limit increment.
        min_limit=value        Set the minumum data limit.
        max_limit=value        Set the maximum data limit.
        maxdata=value          The maximum data limit (all files).
        maxdatap=value         The maximum data percentage (range: 0-100).
        flags=flags            Set open flags:   {excl,sync,...}
        oflags=flags           Set output flags: {append,trunc,...}
        vflags=flags           Set/clear btag verify flags. {lba,offset,...}
        maxbad=value           Set maximum bad blocks to display. (Default: 25)
        onerr=action           Set error action: {abort, continue, or pause}.
        nice=value             Apply the nice value to alter our priority.
        noprogt=value          Set the no progress time (in seconds).
        noprogtt=value         Set the no progress trigger time (secs).
        notime=optype          Disable timing of specified operation type.
        parity=string          Set parity to: {even, odd, or none}.
        pass_cmd=string        The per pass command to execute.
        passes=value           The number of passes to perform.
        pattern=value          The 32 bit hex data pattern to use.
    or  pattern=iot            Use DJ's IOT test pattern.
    or  pattern=incr           Use an incrementing data pattern.
    or  pattern=string         The string to use for the data pattern.
        position=offset        Position to offset before testing.
        oposition=offset       The output file position (copy/verify).
        prefix=string          The data pattern prefix string.
        procs=value            The number of processes to create.
        ralign=value           The random I/O offset alignment.
        rlimit=value           The random I/O data byte limit.
        rseed=value            The random number generator seed.
        records=value          The number of records to process.
        readp=value            Percentage of accesses that are reads. Range [0,100].
                               'random' keyword makes the read/write percentage random.
        randp=value            Percentage of accesses that are random. Range [0,100].
                               Sequential accesses = 0%, else random percentage
        rrandp=value           Percentage of read accesses that are random. Range
[0,100].
        wrandp=value           Percentage of write accesses that are random. Range
[0,100].
```

```
    runtime=time          The number of seconds to execute.
    script=filename       The script file name to execute.
    slices=value          The number of disk slices.
    slice=value           Choose a specific disk slice.
    soffset=value         The starting slice offset.
    skip=value            The number of records to skip past.
    seek=value            The number of records to seek past.
    step=value            The number of bytes seeked after I/O.
    stats=level           The stats level: {brief, full, or none}
    stopon=filename       Watch for file existence, then stop.
    sleep=time            The sleep time (in seconds).
    msleep=value          The msleep time (in milliseconds).
    usleep=value          The usleep time (in microseconds).
    showbtags opts...     Show block tags and btag data.
    showfslba             Show file system offset to physical LBA.
    showfsmap             Show file system map extent information.
    showtime=value        Show time value in ctime() format.
    showvflags=value      Show block tag verify flags set.
    threads=value         The number of threads to execute.
    trigger={br, bdr, lr, seek, cdb:bytes, cmd:str, and/or triage}
                          The triggers to execute on errors.
    trigger_action=value  The trigger action (for noprogs).
    trigger_on={all, errors, miscompare, or noprogs} (Default: all)
                          The trigger control (when to execute).
    volumes=value         The number of volumes to process.
    vrecords=value        The record limit for the last volume.
    enable=flag           Enable one or more of the flags below.
    disable=flag          Disable one or more of the flags below.
    help                  Display this help text.
    eval EXPR             Evaluate expression, show values.
    system CMD            Execute a system command.
    !CMD                  Same as above, short hand.
    shell                 Startup a system shell.
    usage                 Display the program usage.
    version               Display the version information.

I/O Behaviors:
    iobehavior=type       Specify the I/O behavior. (alias: iob=)
      Where type is:
        dt                The dt I/O behavior (default).
        dtapp             The dtapp I/O behavior.
        hammer            The hammer I/O behavior.
        sio               The simple I/O (sio) behavior.

For help on each I/O behavior use: "iobehavior=type help"

Block Tag Verify Flags: (prefix with ~ to clear flag)
    lba,offset,devid,inode,serial,hostname,signature,version
    pattern_type,flags,write_start,write_secs,write_usecs,
    pattern,generation,process_id,thread_number,device_size
    record_index,record_size,record_number,step_offset,
    opaque_data_type,opaque_data_size,crc32

    default Disk: lba,devid,serial + common
    default File: offset,inode + common flags
    common Flags: hostname,signature,write_start,generation,
                  prcoess_id,job_id,thread_number,crc32

    Example: verifyFlags= or vflags=~all,lba,crc32

Force Corruption Options:
    corrupt_index=value   The corruption index. (Default: random)
    corrupt_length=value  The corruption length. (Default: 4 bytes)
    corrupt_pattern=value The corruption pattern. (Default: 0xfeedface)
    corrupt_step=value    Corruption buffer step. (Default: 0 bytes)
    corrupt_reads=value   Corrupt at read records. (Default: 13)
    corrupt_writes=value  Corrupt at write records. (Default: 0)

Job Start Options:
    istate={paused,running} (Default: running)
                          Initial state after thread created.
    tag=string            Specify job tag when starting tests.

Job Control Options:
    jobs[:full][={jid|tag}] | [job=value] | [tag=string]
                          Show all jobs or specified job.
    cancelall             Cancel all jobs.
    cancel={jid|tag} | [job=value] | [tag=string]
                          Cancel the specified job ID.
    modify[={jid|tag}] | [job=value] | [tag=string] [modify_options]
                          Modify all jobs or specified job.
    pause[={jid|tag}] | [job=value] | [tag=string]
                          Pause all jobs or specified job.
    query[={jid|tag}] | [job=value] | [tag=string] [query_string]
                          Query all jobs or specified job.
    resume[={jid|tag}] | [job=value] | [tag=string]
                          Resume all jobs or specified job.
    stopall               Stop all jobs.
    stop={jid|tag} | [job=value] | [tag=string]
                          Stop the specified job.
    wait[={jid|tag}] | [job=value] | [tag=string]
                          Wait for all jobs or specified job.

File System Map Format:
    showfslba [bs=value] [count=value] [limit=value] [offset=value]
                          Show FS offset(s) mapped to physical LBA(s)
                          The default is to show LBA for specified offset.
    showfsmap [bs=value] [count=value] [limit=value] [offset=value]
                          Show the file system map extent information.
                          The default is to show the full extent map.

File Locking Options:
    enable=lockfiles      Enables file locks (locks & unlocks)
    lockmode={mixed | full | partial}
                          Chance of full or partial file locks (default: mixed).
    unlockchance=[0-100]  Probability of keeping locks and skipping unlocking.
    Examples:
        unlockchance=100  100% chance of unlocking, ALL files unlocked. [default]
        unlockchance=50    50% chance of unlocking each file.
        unlockchance=0      0% chance of unlocking, NO files are unlocked.

Workload Options:
    define workloadName options...
                          Define a workload with options.
    workloads [substr]
                          Display the valid workloads.
    workload=name         Select the specified workload.

File System Full Options:
    fsfree_delay=value    FS free space sleep delay.    (Def: 3 secs)
    fsfree_retries=value  FS free space wait retries.   (Def: 10)

    Please consider adding the truncate flag or enable=deleteperpass,
    to free space between passes or with multiple threads to same FS.

Retry Related Options:
        retry_error=value     The error code to retry.
        retry_delay=value     The retry delay.              (Def: 5 secs)
        retry_limit=value     The retry limit.              (Def: 60)
        retryDC_delay=value   The retry corruptions delay.  (Def: 5)
        retryDC_limit=value   The retry corruptions limit.  (Def: 1)

    Error Strings Accepted:
        EIO (5), ENXIO (6), EBUSY (16), ENODEV (19), ENOSPC (28), ESTALE (116)
            OR
        retry_error='*' or -1 to retry all errors.

    SCSI Specific Options:
        idt=string            The Inquiry device type. (both, device, or serial)
        spt_path=string       Path to SCSI (spt) tool.
        spt_options=string    Additional spt options.
        readtype=string       The SCSI read type (read8, read10, read16).
        writetype=string      The SCSI write type (write8, write10, write16,
writev16).
        scsi_recovery_delay=value The SCSI recovery delay.  (Def: 2 secs)
        scsi_recovery_retries=value The SCSI recovery retries.(Def: 60)
        scsi_timeout=value    The SCSI timeout (in ms).     (Def: 0ms)
        unmap_freq=value      The SCSI unmap frequency.     (Def: 0)
        unmap=type            The SCSI unmap type.
         Valid types are: random, unmap, write_same, zerorod.

    Flags to enable/disable:
        aio              POSIX Asynchronous I/O.    (Default: disabled)
        async            Asynchronous job control.  (Default: disabled)
        btags            Block tag control flag.    (Default: disabled)
        compare          Data comparison flag.      (Default: enabled)
        xcompare         Extra btag prefix compare. (Default: disabled)
        coredump         Core dump on errors.       (Default: disabled)
        deleteerrorlog   Delete error log file.     (Default: enabled)
        deleteperpass    Delete files per pass.     (Default: disabled)
        debug            Debug output.              (Default: disabled)
        Debug            Verbose debug output.      (Default: disabled)
        btag_debug       Block tag (btag) debug.    (Default: disabled)
        edebug           End of file debug.         (Default: disabled)
        fdebug           File operations debug.     (Default: disabled)
        jdebug           Job control debug.         (Default: disabled)
        ldebug           File locking debug.        (Default: disabled)
        mdebug           Memory related debug.      (Default: disabled)
        mntdebug         Mount device lookup debug. (Default: disabled)
        pdebug           Process related debug.     (Default: disabled)
        rdebug           Random debug output.       (Default: disabled)
        tdebug           Thread debug output.       (Default: disabled)
        timerdebug       Timer debug output.        (Default: disabled)
        dump             Dump data buffer.          (Default: enabled)
        dumpall          Dump all blocks.           (Default: disabled)
        dump_btags       Dump block tags (btags).   (Default: disabled)
        dump_context     Dump good context block.   (Default: enabled)
        errors           Report errors flag.        (Default: disabled)
        xerrors          Report extended errors.    (Default: enabled)
        eof              EOF/EOM exit status.       (Default: disabled)
        fileperthread    File per thread.           (Default: enabled)
        fill_always      Always fill files.         (Default: disabled)
        fill_once        Fill the file once.        (Default: disabled)
        fsalign          File system align.         (Default: disabled)
        fsmap            File system map control.   (Default: enabled)
        fstrim           File system trim.          (Default: disabled)
        funique          Unique output file.        (Default: disabled)
        fsincr           File size incrementing.    (Default: disabled)
        fsync            Controls file sync'ing.    (Default: runtime)
        header           Log file header.           (Default: disabled)
        trailer          Log file trailer.          (Default: enabled)
        force-corruption Force a FALSE corruption.  (Default: disabled)
        hdump            History dump.              (Default: disabled)
        htiming          History timing.            (Default: disabled)
        image            Image mode copy (disks).   (Default: disabled)
        iolock           I/O lock control.          (Default: disabled)
        lbdata           Logical block data.        (Default: disabled)
        logpid           Log process ID.            (Default: disabled)
        lockfiles        Lock files.                (Default: disabled)
        looponerror      Loop on error.             (Default: disabled)
        microdelay       Microsecond delays.        (Default: disabled)
        msecsdelay       Millisecond delays.        (Default: disabled)
        secsdelay        Second delays.             (Default: disabled)
        mmap             Memory mapped I/O.         (Default: disabled)
        mount_lookup     Mount device lookup.       (Default: enabled)
        multi            Multiple volumes.          (Default: disabled)
        noprog           No progress check.         (Default: disabled)
        pipes            Pipe mode control flag.    (Default: disabled)
        poison           Poison read buffer flag.   (Default: disabled)
        prefill          Prefill read buffer flag.  (Default: runtime)
        job_stats        The job statistics flag.   (Default: disabled)
        pstats           The per pass statistics.   (Default: enabled)
        total_stats      The total statistics.      (Default: enabled)
        raw              Read after write.          (Default: disabled)
        reread           Re-read after raw.         (Default: disabled)
        resfsfull        Restart file system full.  (Default: enabled)
        readcache        Read cache control.        (Default: enabled)
        writecache       Write cache control.       (Default: enabled)
        retryDC          Retry data corruptions.    (Default: enabled)
        retrydisc        Retry session disconnects. (Default: disabled)
        retrywarn        Retry logged as warning.   (Default: disabled)
        savecorrupted    Save corrupted data.       (Default: enabled)
        scriptverify     Script verify display.     (Default: disabled)
        sighup           Hangup signal control.     (Default: enabled)
        nvme_io          NVMe I/O operations.       (Default: disabled)
        scsi             SCSI operations.           (Default: enabled)
        scsi_info        SCSI information.          (Default: enabled)
        scsi_io          SCSI I/O operations.       (Default: disabled)
        sdebug           SCSI debug output.         (Default: disabled)
        scsi_errors      SCSI error logging.        (Default: disabled)
        scsi_recovery    SCSI recovery control.     (Default: enabled)
        unmap            SCSI unmap per pass.       (Default: disabled)
        get_lba_status   SCSI Get LBA Status.       (Default: disabled)
        fua              SCSI Force unit access.    (Default: disabled)
        dpo              SCSI Disable page out.     (Default: disabled)
        stats            Display statistics.        (Default: enabled)
        stopimmed        Stop immediate w/stop file.(Default: enabled)
        syslog           Log errors to syslog.      (Default: disabled)
        terminate_on_signals Terminate on signals.  (Default: disabled)
        timestamp        Timestamp each block.      (Default: disabled)
        trigargs         Trigger cmd arguments.     (Default: enabled)
        trigdefaults     Automatic trigger defaults.(Default: enabled)
        trigdelay        Delay mismatch triggers.   (Default: enabled)
        unique           Unique pattern.            (Default: enabled)
        uuid_dashes      Dashes in UUID strings.    (Default: enabled)
        verbose          Verbose output.            (Default: enabled)
        verify           Verify data written.       (Default: enabled)

        Example: enable=debug disable=compare,pstats

    Common Open Flags:
        none                  Clear all user set flags.
        excl (O_EXCL)         Exclusive open. (don't share)
        ndelay (O_NDELAY)     Non-delay open. (don't block)
        nonblock (O_NONBLOCK) Non-blocking open/read/write.
        direct (O_DIRECT)     Direct disk access. (don't cache data).
        nodirect              Cache data (disables Direct I/O).
        fsync (O_FSYNC)       Sync both read/write data with disk file.
        rsync (O_RSYNC)       Synchronize read operations.
        sync (O_SYNC)         Sync updates for data/file attributes.
        large (O_LARGEFILE)   Enable large (64-bit) file system support.

    Output Open Flags:
        none                  Clear all user set flags.
        append (O_APPEND)     Append data to end of existing file.
        dsync (O_DSYNC)       Sync data to disk during write operations.
        trunc (O_TRUNC)       Truncate an existing file before writing.

    Delays (Values are seconds, unless micro/msecs delay is enabled):
        open_delay=value      Delay before opening the file.    (Default: 0)
        close_delay=value     Delay before closing the file.    (Default: 0)
        delete_delay=value    Delay after deleting files.       (Default: 0 secs)
        end_delay=value       Delay between multiple passes.    (Default: 0 secs)
        forced_delay=value    Force random I/O delay (noprog).  (Default: 0 secs)
        start_delay=value     Delay before starting the test.   (Default: 0 secs)
        read_delay=value      Delay before reading each record. (Default: 0)
        verify_delay=value    Delay before verifying data.      (Default: 0)
        write_delay=value     Delay before writing each record. (Default: 0)
        term_delay=value      Delay before terminating.         (Default: 0 secs)
        term_wait=time        Thread termination wait time.     (Default: 180 secs)

        The delay options accept 'random' for random delays.
        Please Note: For disk devices, microseconds is the default!:

    Numeric Input:
        For options accepting numeric input, the string may contain any
        combination of the following characters:

        Special Characters:
            w = words (4 bytes)            q = quadwords (8 bytes)
            b = blocks (512 bytes)         k = kilobytes (1024 bytes)
            m = megabytes (1048576 bytes)  p = page size (4096 bytes)
            g = gigabytes (1073741824 bytes)
            t = terabytes (1099511627776 bytes)
            d = device size (set via dsize=value option)
            inf or INF = infinity (18446744073709551615 bytes)

        Arithmetic Characters:
            + = addition                   - = subtraction
            * or x = multiplcation         / = division
            % = remainder

        Bitwise Characters:
            ~ = complement of value       >> = shift bits right
           << = shift bits left            & = bitwise 'and' operation
            | = bitwise 'or' operation     ^ = bitwise exclusive 'or'

        The default base for numeric input is decimal, but you can override
        this default by specifying 0x or 0X for hexadecimal conversions, or
        a leading zero '0' for octal conversions.  NOTE: Evaluation is from
        right to left without precedence, and parenthesis are not permitted.

    Keepalive Format Control:
            %b = The bytes read or written.   %B = Total bytes read and written.
            %c = Record count for this pass.  %C = Total records for this test.
            %d = The device/file name.         %D = The real device name.
            %e = The number of errors.         %E = The error limit.
            %f = The files read or written.    %F = Total files read and written.
            %h = The host name.                %H = The full host name.
            %k = The kilobytes this pass.      %K = Total kilobytes for this test.
            %l = Blocks read or written.       %L = Total blocks read and written.
            %m = The megabytes this pass.      %M = Total megabytes for this test.
            %p = The pass count.               %P = The pass limit.
            %r = Records read this pass.       %R = Total records read this test.
            %s = The seconds this pass.        %S = The total seconds this test.
            %t = The pass elapsed time.        %T = The total elapsed time.
            %i = The I/O mode (read/write)     %u = The user (login) name.
            %w = Records written this pass.    %W = Total records written this test.

    Performance Keywords:
            %bps  = The bytes per second.      %lbps = Logical blocks per second.
            %kbps = Kilobytes per second.      %mbps = The megabytes per second.
            %iops = The I/O's per second.      %spio = The seconds per I/O.

            Lowercase means per pass stats, while uppercase means total stats.

    I/O Keywords:
            %iodir = The I/O direction.        %iotype = The I/O type.
            %lba = The current logical block. %offset = The current file offset.
            %elba = The error logical block.  %eoffset = The error file offset.
            %bufmode = The file buffer mode.  %status = The thread exit status.

    Job Control Keywords:
            %job  = The job ID.                %tag    = The job tag.
            %tid  = The thread ID.             %thread = The thread number.

    Misc Keywords:
            %keepalivet = The keepalive time.

    Default Keepalive:
            keepalive="%d Stats: mode %i, blocks %l, %m Mbytes, pass %p/%P, elapsed
%t"

    Default Pass Keepalive: (when full pass stats are disabled via disable=pstats)
            pkeepalive="%d Stats: mode %i, blocks %L, %M Mbytes, pass %p/%P, elapsed
%T"

    Common Format Control Keywords:
            %array   = The array name or management IP.
            %bufmode = The file buffer mode.  %dfs    = The directory separator ('/')
            %dsf     = The device name.       %device = The device path.
            %sdsf    = The SCSI device name.  %tdsf   = The trigger device name.
            %file    = The file name.         %devnum = The device number.
            %host    = The host name.         %user   = The user name.
            %job     = The job ID.            %tag    = The job tag.
            %jlog    = The job log.           %tlog   = The Thread log.
            %tid     = The thread ID.         %thread = The thread number.
            %pid     = The process ID.        %prog   = The program name.
            %ymd     = The year,month,day.    %hms    = The hour,day,seconds.
            %date    = The date string.       %et     = The elapsed time.
            %tod     = The time of day.       %etod   = Elapsed time of day.
            %secs    = Seconds since start.   %seq    = The sequence number.
            %script  = The script file name.  %tmpdir = The temporary directory.
            %uuid    = The UUID string.       %workload = The workload name.
            %month   = The month of the year. %day    = The day of the month.
            %year    = The four digit year.   %hour   = The hour of the day.
            %minutes = The minutes of hour.   %seconds= The seconds of minute.
            %nate    = The NATE log prefix.   %nos    = The Nimble log prefix.
            String 'gtod' = "%tod (%etod) %prog (j:%job t:%thread): "

        Example: log=dt_%host_%user_%iodir_%iotype-%uuid.log
                 logprefix="%seq %ymd %hms %et %prog (j:%job t:%thread): "

    SCSI Format Keywords:
            %capacity = The disk capacity.    %blocklen = The disk block length.
            %vendor = The Inquiry vendor ID.  %product = The Inquiry product ID.
            %revision = The Inquiry revision. %devid = The device identifier.
            %serial = The disk serial number. %mgmtaddr = The management address.

    I/O Tune File Format Keywords:
            %iotune = The I/O tune path.      %tmpdir = The temporary directory.
            %host   = The host name.          %user   = The user (login) name.

        Example: iotune=%iotune OR %tmpdir%host_MyIOtune_file

    Pattern String Input:
            \\ = Backslash   \a = Alert (bell)   \b = Backspace
            \f = Formfeed    \n = Newline        \r = Carriage Return
            \t = Tab         \v = Vertical Tab   \e or \E = Escape
            \ddd = Octal Value    \xdd or \Xdd = Hexadecimal Value

    Prefix Format Control:
            %d = The device/file name.      %D = The real device name.
            %h = The host name.             %H = The full host name.
            %p = The process ID.            %P = The parent PID.
            %s = The device serial number.
            %u = The user name.             %U = A unique UUID.
            %j = The job ID.                %J = The job tag.
            %t = The thread number.         %T = The thread ID.

        Example: prefix="%U %d@%h" OR prefix="%d(%s)@%h"

    Time Input:
            d = days (86400 seconds),      h = hours (3600 seconds)
            m = minutes (60 seconds),      s = seconds (the default)

        Arithmetic characters are permitted, and implicit addition is
        performed on strings of the form '1d5h10m30s'.

    Trigger Types:
            br = Execute a bus reset.
            bdr = Execute a bus device reset.
            lr = Execute a device lun reset.
            seek = Issue a seek to the failing lba.
            triage = Issue SCSI triage commands.
            cmd:string = Execute command with these args:
              string dname op dsize offset position lba errno noprogt
              args following cmd:string get appended to above args.

    Defaults:
        errors=1, files=0, passes=1, records=0, bs=512, log=stderr
        pattern=0x39c39c39, dispose=delete, align=0 (page aligned)
        aios=8, dlimit=512, onerr=continue, volumes=0, vrecords=1
        iodir=forward, iomode=test, iotype=sequential, stats=full
        iotseed=0x01010101, hdsize=32, maxbad=25

    --> Date: September 21st, 2023, Version: 25.05, Author: Robin T. Miller <--
```

Appendix B Test Strategy

Depending on your needs, *dt* provides a wide range of options to help customize your tests. My preference is to use a variety of tests, and for that matter different test tools, since each tool has its' own strengths and I/O patterns. But in general, *dt* serves most of my needs. Here are a couple things to keep in mind while developing your test strategy:

- Are you testing storage, driver, firmware, switch, file systems, network, or all?
  - what are the buffer alignment restrictions (if any)?
  - what are the characteristics of the component (s)?
  - what are the debug capabilities (for trouble-shooting)?
  - what mechanisms are available to stop I/O on errors?
  - what is the best trigger mechanism? consider *scu* or *spt* if SCSI.
  - what tunables are available? queue depth, max transfer size, etc.
- What are your testing goals: stress testing, or reliability testing?
  - consider using a wide variety of variable request sizes.
  - consider using *runtime*= option to specify length of test times.
  - consider using *errors*= option to tolerate a number of errors.
- Are you concerned with buffer alignment and/or pattern sensitive data?
  - consider using *align*=, *pattern*=, and *pf*= options.
- Are you doing shared (multi-initiatior) style storage testing?
  - consider using *slices*= and *slice*= options to test sections of disks simultaneously from each host (an integral and necessary part of shared storage testing)
  - consider using *prefix*= to create unique string from each host.
- Before generating an I/O load, please consider the following:
  - what is the service queue limits of your storage (max I/O requests)?
  - what is the queue depth of your storage device, driver, and host adapter?
  - does your host OS disk driver handle "queue fulls" well?
  - how many hosts are accessing your shared storage?
  - how many processors, memory, and swap space is available?
  - depending out the above, you may need to limit your I/O loads.
- How much I/O load do you wish to generate?
  - consider *aios*=, *procs*=, and/or *slices*= options.
  - don't overdrive your host OS or storage (unless intended).
  - don't spawn so many processes that paging/swapping occurs.
- What tools are available for monitoring the I/O load?
  - consider using *iostat*, *vmstst*, *top*, etc to monitor I/O and processes.
  - consider monitoring per path I/O, e.g. AIX "*iostat –m*" w/MPIO.
  - consider monitoring/gathering statistics from your storage array.
  - consider using *scu* to gather SCSI Log Page statistics.
- Are you doing perturbation testing?
  - abort, bus/target/lun resets? consider using *scu* or *spt.*
  - do you need to do panic and reboot testing?

  - will you be doing storage controller failures?
  - consider failover characteristics of your storage.
  - consider using *alarm*= and *noprogt*= options to monitor I/O.
- Consider tools for troubleshooting problems:
  - does the OS supply an error logger? (AIX has *errpt*).
  - where do kernel error messages get written?
  - can you display kernel messages via *dmesg*?
  - does your host supply a method to panic the system?
  - consider using *trigger*= option to trigger analyzers or stop software traces.
- Are you doing performance testing?
  - use *aios=value* option with large value (say 64).
  - use larger block sizes: e.g. *bs=64k to 256k* or greater.
  - disable data comparisions via *disable=compar.*
  - keep buffers page aligned (i.e., don't use *align*= option).
  - do **not** use read-after-write (*enable=raw*) option.
  - sequntial I/O is always faster than random I/O (of course).
  - during file system testing, umount/re-mount to invalidate the buffer cache.
  - keep in mind, *dt* was not developed to be a perfomance tool (though useful).

Obviously, this is only some of what needs to be considered. Each storage device, host OS, drivers, etc. have different attributes. Each lab has their own requirements, and *dt* is usually wrapped by some test harness or I/O manager automation. Sadly, none of these are open sourced nor productized for purchase, so test harnesses or scripts need to be developed (today).

Appendix C *dt* Workloads

This section contains various *dt* examples used to show its' capabilities and to help get new users started. A short description prefaces each test to describe the nature of the test being performed.

Note: If you are new to *dt*, please review Documentation/dt-ToolOverview.pdf for an introduction. There are also several other useful files in the Documentation directory, as well as useful scripts in the Scripts directory that can help guide your initial footsteps.

The examples below will mainly utilize *dt*'s pre-defined workloads. These workloads are known to work and provide sets of options and/or flags that are not easily understood by novice users.

Reviewing Workloads

To see all the pre-defined workloads, use "**dt workloads**". I have not included all the workloads here, since that will take several pages. Instead, we will look at specific workloads and examples.

First, we will review the longevity workloads available:

```
$ dt workloads longevity
Valid Workloads:

    longevity_common: Longevity Common Options (template)
        min=8k max=1m incr=vary enable=raw,reread,log_trailer,syslog
history=5 history_data=152 enable=history_timing logprefix='%seq %nos %et
%prog (j:%job t:%thread): ' keepalivet=5m runtime=-1 onerr=abort noprogt=30s
noprogtt=5m stopon=C:\temp\stopit

    longevity_file_dedup: Longevity File System w/Dedup Workload
        workload=longevity_common min_limit=1m max_limit=2g incr_limit=vary
dispose=keep flags=direct notime=close,fsync oflags=trunc maxdatap=75
threads=4 pf=x:\noarch\dtdata\pattern_dedup

    longevity_disk_dedup: Longevity Direct Disk w/Dedup Workload
        workload=longevity_common capacityp=75 slices=4
pf=x:\noarch\dtdata\pattern_dedup

    longevity_file_system: Longevity File System Workload
        workload=longevity_common workload=high_validation min_limit=1m
max_limit=2g incr_limit=vary dispose=keep flags=direct notime=close,fsync
oflags=trunc maxdatap=75 threads=4

    longevity_disk_unmap: Longevity Direct Disk w/SCSI UNMAP Workload
        workload=longevity_common workload=high_validation capacityp=75
slices=4 unmap=unmap

    longevity_disk: Longevity Direct Disk Workload
```

```
        workload=longevity_common workload=high_validation capacityp=75
slices=4

    longevity_disk_write_only: Longevity Direct Disk Write Only
        workload=longevity_disk disable=raw,reread,verify

    longevity_file_write_only: Longevity File System Write Only
        workload=longevity_file_system disable=raw,reread,verify

$ dt workload high_validation
Valid Workloads:

    high_validation: Define Highest Data Validation Options (template)
        enable=btags pattern=iot prefix='%d@%h'

$
```

You'll notice workloads can include workload templates which as common options above. The data files, used with the pattern file option above, are found in the data/ directory.

Also, details on compression and deduplication can be found in this file Documents/ 'dt+Compression+&+Deduplication.doc'.

The workloads that include "disk" are for direct disk testing, while those that include "file" are file system tests. Please note options that include a path, will need to be updated for your hosts. Input and output options are *not* included in workload definitions, so *must* be specified on the command line, since these vary (of course).

Options can be overridden, so oftentimes it's useful to start with a particular workload, then add or override options desired.

Executing Workloads

This example is to my Windows system disk, a PCI NVME SSD, thus high performance:

```
$ rm -rf dtfiles
$ dt of=dtfiles/dt.data workload=longevity_file_system runtime=30s logprefix="%prog (j:%job
t:%thread): "
dt.exe (j:0 t:0): Warning: Top level directory dtfiles, will *not* be deleted!
dt.exe (j:1 t:1):
dt.exe (j:1 t:1): Read After Write Statistics:
dt.exe (j:1 t:1):            Current random seed: 0x36871875132
dt.exe (j:1 t:1):       Job Information Reported: Job 1, Thread 1
dt.exe (j:1 t:1):        Current Thread Reported: 1/4
dt.exe (j:1 t:1):       Last IOT seed value used: 0x01010101
dt.exe (j:1 t:1):         Block tag verify flags: 0x0fdff7ef
dt.exe (j:1 t:1):        Maximum data calculated: 36659149824 (34960.890 Mbytes, 34.141 Gbytes)
dt.exe (j:1 t:1):        Total records processed: 1208 with min=8192, max=1048576, incr=variable
dt.exe (j:1 t:1):        Total bytes transferred: 653403136 (638089.000 Kbytes, 623.134 Mbytes)
dt.exe (j:1 t:1):         Average transfer rates: 335117542 bytes/sec, 327263.225 Kbytes/sec,
319.593 Mbytes/sec
dt.exe (j:1 t:1):        Number I/O's per second: 619.559
dt.exe (j:1 t:1):         Number seconds per I/O: 0.0016 (1.61ms)
dt.exe (j:1 t:1):         Total passes completed: 0
dt.exe (j:1 t:1):          Total errors detected: 0/1
dt.exe (j:1 t:1):             Total elapsed time: 00m02.00s
dt.exe (j:1 t:1):                  Starting time: Sun Jan 19 16:35:15 2025
dt.exe (j:1 t:1):                    Ending time: Sun Jan 19 16:35:17 2025
```

```
                …
dt.exe (j:1 t:3):
dt.exe (j:1 t:3): Operating System Information:
dt.exe (j:1 t:3):                     Host name: DESKTOP-SBC6MG3 (192.168.50.26)
dt.exe (j:1 t:3):                     User name: robin
dt.exe (j:1 t:3):                    Process ID: 4344
dt.exe (j:1 t:3):                OS information: Windows 10 [10.0.19045 No Service Pack]
dt.exe (j:1 t:3):
dt.exe (j:1 t:3): File System Information:
dt.exe (j:1 t:3):                   Volume name: Windows
dt.exe (j:1 t:3):               Filesystem type: NTFS
dt.exe (j:1 t:3):         Filesystem block size: 4096
dt.exe (j:1 t:3):         Filesystem free space: 191323377664 (182460.191 Mbytes, 178.184
Gbytes, 0.174 Tbytes)
dt.exe (j:1 t:3):        Filesystem total space: 510770802688 (487108.996 Mbytes, 475.692
Gbytes, 0.465 Tbytes)
dt.exe (j:1 t:3):              Volume path name: C:\
dt.exe (j:1 t:3):          Volume serial number: 1690051761
dt.exe (j:1 t:3):
dt.exe (j:1 t:3): Total Statistics:
dt.exe (j:1 t:3):       Output device/file name: dtfiles/dt.data-j1t3 (device type=regular)
dt.exe (j:1 t:3):       Type of I/O's performed: sequential (forward, rseed=0x36881ad153e, read-
after-write)
dt.exe (j:1 t:3):      Job Information Reported: Job 1, Thread 3
dt.exe (j:1 t:3):       Current Thread Reported: 3/4
dt.exe (j:1 t:3):      Data pattern prefix used: dtfiles/dt.data-j1t3@DESKTOP-SBC6MG3
dt.exe (j:1 t:3):      Data pattern string used: 'IOT Pattern' (blocking is 512 bytes)
dt.exe (j:1 t:3):      Last IOT seed value used: 0x03030303
dt.exe (j:1 t:3):        Block tag verify flags: 0x0fdff7ef
dt.exe (j:1 t:3):       Maximum data calculated: 36659149824 (34960.890 Mbytes, 34.141 Gbytes)
dt.exe (j:1 t:3):            Total records read: 6704
dt.exe (j:1 t:3):              Total bytes read: 3600864256 (3516469.000 Kbytes, 3434.052
Mbytes, 3.354 Gbytes)
dt.exe (j:1 t:3):         Total records written: 3444
dt.exe (j:1 t:3):           Total bytes written: 1849401344 (1806056.000 Kbytes, 1763.727
Mbytes, 1.722 Gbytes)
dt.exe (j:1 t:3):       Total records processed: 10148 with min=8192, max=1048576, incr=variable
dt.exe (j:1 t:3):       Total bytes transferred: 5450265600 (5322525.000 Kbytes, 5197.778
Mbytes)
dt.exe (j:1 t:3):        Average transfer rates: 184877111 bytes/sec, 180544.054 Kbytes/sec,
176.313 Mbytes/sec
dt.exe (j:1 t:3):       Number I/O's per second: 344.228
dt.exe (j:1 t:3):        Number seconds per I/O: 0.0029 (2.91ms)
dt.exe (j:1 t:3):        Total passes completed: 2
dt.exe (j:1 t:3):         Total errors detected: 0/1
dt.exe (j:1 t:3):            Total elapsed time: 00m30.00s
dt.exe (j:1 t:3):                 Starting time: Sun Jan 19 16:35:15 2025
dt.exe (j:1 t:3):                   Ending time: Sun Jan 19 16:35:45 2025
dt.exe (j:1 t:3):
dt.exe (j:1 t:3): Command line to re-read the data:
dt.exe (j:1 t:3):     % dt.exe if=dtfiles/dt.data-j1t3 min=8192 max=1048576 incr=vary dsize=512
iotype=sequential iodir=forward limit=97938432 records=184 rseed=0x36881ad153e
prefix="dtfiles/dt.data-j1t3@DESKTOP-SBC6MG3" pattern=iot iotseed=0x03030303 enable=btags
flags=direct
dt.exe (j:1 t:3):
dt.exe (j:1 t:3): Command Line:
dt.exe (j:1 t:3):
dt.exe (j:1 t:3):     % dt.exe of=dtfiles/dt.data workload=longevity_file_system runtime=30s
logprefix="%prog (j:%job t:%thread): "
dt.exe (j:1 t:3):
dt.exe (j:1 t:3):        --> Date: September 21st, 2023, Version: 25.05, Author: Robin T. Miller
<--
dt.exe (j:1 t:3):

$
```

Defining Workloads

The “*define*” command can be used to add your own workloads. Using the above command, we will add this to a new workload by its’ own name via the *dt* startup file:

```
$ cat >~/.datatestrc

define myworkload: workload=longevity_file_system runtime=30s
logprefix="%prog (j:%job t:%thread): "

<Ctrl/D>

$ dt workloads myworkload

Valid Workloads:

    myworkload:

        workload=longevity_file_system runtime=30s logprefix="%prog (j:%job
t:%thread): "

$ dt of=dtfiles/dt.data workload=myworkload
```

The *dt* startup file is expected to be in the user home directory.

Since *dt* workloads can be complex, you may find this feature useful to simplify workloads. 😊

Appendix D Block Tags

Block tags provide unique data at the beginning of each data block. It creates unique data per block and provides information to help troubleshoot data corruptions (aka miscompares).

Please Note: Block tags do require more compute power due to filling and CRC generation.

Here’s an example of block tags and what they look like:

```
$ dt of=dtbtags.data bs=random enable=btags records=10 dispose=keep disable=stats,trailer,verbose

$ dt if=dtbtags.data showbtag offset=25k count=1 logprefix="" disable=trailer

Block Tag (btag) @ 0x0000000000582000 (152 bytes):

             File Offset (  0): 25600 (0x6400)
                 File ID (  8): 5348024557687781 (0x1300000002d3e5)
               Host Name ( 52): DESKTOP-SBC6MG3
               Signature ( 76): 0xbadcafee
                 Version ( 80): 1
            Pattern Type ( 81): 3 (32-bit pattern)
                   Flags ( 82): 0x1 (file,sequential,forward)
 Write Pass Start (secs) ( 84): 0x678fc6bb => Tue Jan 21 16:09:31 2025
  Write Timestamp (secs) ( 88): 0x678fc6bb => Tue Jan 21 16:09:31 2025
 Write Timestamp (usecs) ( 92): 0x000367f8
                 Pattern ( 96): 0x39c39c39
              Generation (100): 1 (0x00000001)
              Process ID (104): 16484 (0x00004064)
                  Job ID (108): 1 (0x00000001)
           Thread Number (112): 1 (0x00000001)
             Device Size (116): 512 (0x00000200)
            Record Index (120): 25600 (0x00006400)
             Record Size (124): 641536 (0x0009ca00)
           Record Number (128): 1 (0x00000001)
             Step Offset (136): 0 (0x0)
        Opaque Data Type (144): 0 (No Data Type)
        Opaque Data Size (146): 0 (0x0000)
                  CRC-32 (148): 0x49c41617

Dumping Block Tag File offsets (base offset = 25600, limit = 152 bytes):
                  / Block
      File Offset / Index
0000000000025600/     0 | 00 64 00 00 00 00 00 00 e5 d3 02 00 00 00 13 00 " d              "
0000000000025616/    16 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "                "
0000000000025632/    32 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "                "
0000000000025648/    48 | 00 00 00 00 44 45 53 4b 54 4f 50 2d 53 42 43 36 "    DESKTOP-SBC6"
0000000000025664/    64 | 4d 47 33 00 00 00 00 00 00 00 00 00 ee af dc ba "MG3             "
0000000000025680/    80 | 01 03 01 00 bb c6 8f 67 bb c6 8f 67 f8 67 03 00 "       g   g g  "
0000000000025696/    96 | 39 9c c3 39 01 00 00 00 64 40 00 00 01 00 00 00 "9  9    d@      "
0000000000025712/   112 | 01 00 00 00 00 02 00 00 00 64 00 00 00 ca 09 00 "         d      "
0000000000025728/   128 | 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "                "
0000000000025744/   144 | 00 00 00 00 17 16 c4 49                         "       I"

$
```

Appendix E *IOT Pattern*

The IOT pattern creates unique data per data block. The first four bytes is the block number which is then used with a four-byte value to seed the remaining part of the block. This seed is multiplied by the pass count, to ensure that data is unique per pass.
The IOT pattern when used in conjunction with block tags creates the highest level of data validation. This pattern has additional support for corruption analysis, which provides a summary of good and bad blocks and a side-by-side display of corrupted data.

IOT example:

```
$ dt of=dtiot.data pattern=iot records=1 dispose=keep disable=stats,trailer

$ od -Ad -tx4 -w dtiot.data

0000000 00000000 01010101 02020202 03030303 04040404 05050505 06060606 07070707
0000032 08080808 09090909 0a0a0a0a 0b0b0b0b 0c0c0c0c 0d0d0d0d 0e0e0e0e 0f0f0f0f
0000064 10101010 11111111 12121212 13131313 14141414 15151515 16161616 17171717
0000096 18181818 19191919 1a1a1a1a 1b1b1b1b 1c1c1c1c 1d1d1d1d 1e1e1e1e 1f1f1f1f
0000128 20202020 21212121 22222222 23232323 24242424 25252525 26262626 27272727
0000160 28282828 29292929 2a2a2a2a 2b2b2b2b 2c2c2c2c 2d2d2d2d 2e2e2e2e 2f2f2f2f
0000192 30303030 31313131 32323232 33333333 34343434 35353535 36363636 37373737
0000224 38383838 39393939 3a3a3a3a 3b3b3b3b 3c3c3c3c 3d3d3d3d 3e3e3e3e 3f3f3f3f
0000256 40404040 41414141 42424242 43434343 44444444 45454545 46464646 47474747
0000288 48484848 49494949 4a4a4a4a 4b4b4b4b 4c4c4c4c 4d4d4d4d 4e4e4e4e 4f4f4f4f
0000320 50505050 51515151 52525252 53535353 54545454 55555555 56565656 57575757
0000352 58585858 59595959 5a5a5a5a 5b5b5b5b 5c5c5c5c 5d5d5d5d 5e5e5e5e 5f5f5f5f
0000384 60606060 61616161 62626262 63636363 64646464 65656565 66666666 67676767
0000416 68686868 69696969 6a6a6a6a 6b6b6b6b 6c6c6c6c 6d6d6d6d 6e6e6e6e 6f6f6f6f
0000448 70707070 71717171 72727272 73737373 74747474 75757575 76767676 77777777
0000480 78787878 79797979 7a7a7a7a 7b7b7b7b 7c7c7c7c 7d7d7d7d 7e7e7e7e 7f7f7f7f
0000512

$
```

Appendix F *File System Map*

On Linux and Windows, the file system map is used to report physical logical block numbers in extended error reporting as well as corruption analysis. This information is important to product developers for tracing and identifying failures, usually data corruption or an I/O error.
On Windows, you must be running as Administrator, and the file must be fully populated, rather than a sparse file. On Linux, non-root users can see the underlying file mapping.

This is a Windows example, using the previously created block tags file:

```
PS C:\cygwin64\home\robin\dttesting> .\dt.exe if=dtbtags.data showfsmap
dt.exe (j:0 t:0): File: dtbtags.data, LBA Size: 512 bytes
dt.exe (j:0 t:0): Physical Disk: \\.\PhysicalDrive0, Cluster Size: 4096 on \\.\C: [NTFS]
dt.exe (j:0 t:0):
dt.exe (j:0 t:0):    File Offset    Start LBA      End LBA     Blocks      VCN        LCN
dt.exe (j:0 t:0):              0    347169792    347170560        768        0   43325312
dt.exe (j:0 t:0):         393216    357593856    357595136       1280       96   44628320
dt.exe (j:0 t:0):        1048576    367749088    367749216        128      256   45897724
dt.exe (j:0 t:0):        1114112    494971776    494973696       1920      272   61800560
dt.exe (j:0 t:0):        2097152    494975872    494977920       2048      512   61801072
dt.exe (j:0 t:0):        3145728    442878400    442879040        640      768   55288888
dt.exe (j:0 t:0):        3473408    439626016    439626144        128      848   54882340
dt.exe (j:0 t:0):        3538944    437077472    437078752       1280      864   54563772
dt.exe (j:0 t:0):        4194304    433939936    433940320        384     1024   54171580
dt.exe (j:0 t:0):        4390912    389016704    389017984       1280     1072   48556176
dt.exe (j:0 t:0):        5046272    336914720    336916214       1494     1232   42043428
PS C:\cygwin64\home\robin\dttesting>
```

File Documentation/dt-MapFileSystemOffsets.pdf has more details.

Please know that whenever errors or corruptions occur, the file map is used to report both the relative file offset and the physical LBA with extended error reporting. 😊

Appendix G *Trigger Script*

The *trigger=* option allows and external program or script to get control when errors occurs and/or when the no-progress time *noprogt=* option is used. Note, the no-progress trigger time *noprogtt=* option can also be used. This allows you to monitor the no-progress time, then trigger the script at a different (higher) time value.

Below is an example trigger script used on AIX.
Observe the exit status which controls *dt*'s action when this script exits.

```
#!/usr/bin/ksh
# /x/eng/locals/powerpc-ibm-aix5.1/test/bin                    #
# We get called with these parameters                          #
# dname op dsize offset position lba errno noprogtime          #
# Where:                                                       #
#       $1 dname = The device/file name.                       #
#       $2 op = open/close/read/write/miscompare/noprog        #
#       $3 dsize = The device block size.                      #
#       $4 offset = The current file offset.                   #
#       $5 position = The failing offset within block.         #
#       $6 lba = The logical block address (relative for FS). #
#       $7 errno = The error number on syscall errors.         #
#       $8 noprogtime = The no progress time                   #
#                                                              #
# Capture and display these parameters                         #
my_name="dt_io_timeout.ksh"
dev_name=$1
operation=$2
dev_bk_sz=$3
off_set=$4
pos=$5
log_blk=$6
err_num=$7
no_prog_time=$8

echo "$my_name *#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
echo "The device name is $dev_name."
echo "The operation is $operation."
echo "The device block size is $dev_bk_sz."
echo "The offset, position and lba are $off_set, $pos, $log_blk."
echo "The errno is $err_num."
echo "The no progress time is $no_prog_time."
echo "$my_name *#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"

if [[ $operation = "noprog" ]]; then
# return code meanings                                  #
# CONTINUE = 0, TERMINATE = 1, SLEEP = 2, or ABORT = 3 #
        dt_io_timeout_rc=3
        echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
        echo "$my_name: I/O has exceeded the limit."
        echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
        # Now run something useful to display information - like host_info #
        echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
        echo "$my_name: Running host_info"
        echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
        /x/eng/locals/powerpc-ibm-aix5.1/test/bin/host_info
else
   dt_io_timeout_rc=0
fi
echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"
echo "The return code that $my_name is sending to dt is $dt_io_timeout_rc."
echo "*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#"

# Set the return code and exit #
exit $dt_io_timeout_rc
```

Appendix H S3 Cloud *Storage*

Since many folks are moving towards cloud storage, I've added this section.

For testing object storage, I've created two shell scripts.

- Scripts/s3t.sh – main script to generate and test S3 via AWS tools.
- Scripts/test-s3t.sh – wrapper for *s3t.sh* to test multiple buckets.

There are various ways to verify S3 object storage, but the method used by the above script is to use *dt* as a data generation tool to create files on a local file system, upload dt files to cloud storage, download those files, then verify the accuracy of downloaded files using *dt*. This script only took ~4 hours to create and verify! 😉

While a request was made to add AWS API's directly into *dt*, this is an effort I was unprepared to devote time to, and a short term contract was *not* offered. ☹

For this example, please know that my Cloud Storage server is the open source *minio* package, which provides a Amazon Web Service (AWS) set of API's. I then installed AWS tools to utilize in the script. My OS is Ubuntu using Windows Subsystem for Linux (WSL).

My AWS setup looks like this:

```
robin@LAPTOP-BJH5MV95:~$ ls ~/.aws/
config  credentials
robin@LAPTOP-BJH5MV95:~$ cat ~/.aws/config
[default]
s3 =
    signature_version = s3v4
region = us-west-2
endpoint_url = http://172.19.177.60:9000
robin@LAPTOP-BJH5MV95:~$
robin@LAPTOP-BJH5MV95:~$ cat ~/.aws/credentials
[default]
aws_access_key_id = Q6SsJgtbkAmiQqE4qDJM
aws_secret_access_key = tNm8z7UBTIRKzQfSSabM61UZVLOgRZPzpyNYV6UA
robin@LAPTOP-BJH5MV95:~$
```

With credentials already set up, the scripts do not need to specify these credentials.

Copy of *s3t.sh* Script

The s3t.sh script is rather simple and relatively short, so I'm including it here for reference:

```
#!/usr/bin/bash
#
# Date: December 7th, 2023
# Author: Robin T. Miller
#
# Description:
#  Simple script to use dt as a data generation tool for testing S3 object storage.
#  Note: Assumes S3 bucket already exists, and credentials have been created.\
#  This script uses the default profile, aka 's3'.
#
# Modification History:
#   April 19th, 2024
#     Minor updates, fix bash error redirection.
#
#   December 18th, 2023 by Robin T. Miller
#     When reading files, remove the min/max limit options to ensure
#     the whole file is verified, otherwise only a portion is read.
#
#   December 13th, 2023 by Robin T. Miller
#     Add S3 bucket name to dt prefix string.
#     Change the default S3 bucket name to "dt-bucket".
#     If the S3 bucket does not exist, make the bucket.
#
function check_error
{
    exit_status=$?
    if [[ $exit_status -ne 0 ]]; then
        echo "Error occurred, last exit status is ${exit_status}"
        exit $exit_status
    fi
    return
}

# Set defaults:
dtpath=${DTPATH:-~/dt}
bucket=${BUCKET:-dt-bucket}
dtdir=${DTDIR:-dtfiles}
s3dir=${s3DIR:-s3files}
files=${FILES:-10}
limit=${LIMIT:-10m}
passes=${PASSES:-3}
threads=${THREADS:-5}
s3uri="s3://${bucket}"

echo "--> Verify Bucket ${s3uri} Exists <--"
aws s3 ls ${s3uri} 2>/dev/null >/dev/null
if [[ $? -ne 0 ]]; then
    echo "--> Making Bucket ${s3uri} <--"
    aws s3 mb ${s3uri}
    check_error
fi

for pass in $(seq $passes);
do
    echo "--> Starting Pass ${pass} <--"
    echo "--> Removing previous test files <--"
    rm -rf ${dtdir} ${s3dir}
    echo "--> Creating dt files <--"
    ${dtpath}  of=${dtdir}/dt.data bs=random min_limit=4k max_limit=${limit} incr_limit=vary
workload=high_validation threads=${threads} files=${files} dispose=keep iotpass=$pass
disable=pstats prefix="%d@%h@${bucket}"
    check_error
    ls -lsR ${dtdir}
    echo "--> Uploading dt files to S3 server <--"
    aws s3 cp ${dtdir} ${s3uri}/ --recursive
    check_error
    echo "--> Downloading S3 dt files <--"
    aws s3 cp ${s3uri}/ ${s3dir} --recursive
    check_error
    echo "--> Verifying downloaded S3 dt files <--"
    ${dtpath}  if=${s3dir}/dt.data bs=random workload=high_validation vflags=~inode
threads=${threads} files=${files} iotpass=${pass} disable=verbose prefix="%d@%h@${bucket}"
    check_error
    echo "--> Removing S3 dt files <--"
    aws s3 rm --recursive ${s3uri}
    check_error
done
```

Copy of *test-s3t.sh* Script

Here's a copy of the *test-s3t.sh* script, used to test multiple buckets:

```
# Defaults:
# Simple script to start multiple s3t.sh instances.

#bucket=${BUCKET:-dt-bucket}
#dtdir=${DTDIR:-dtfiles}
#s3dir=${s3DIR:-s3files}

unset BUCKET
unset DTDIR
unset s3DIR

for instance in {1..3};
do
    export BUCKET="dt-bucket-${instance}"
    export DTDIR="dtfiles-${instance}"
    export s3DIR="s3files-${instance}"
    ./s3t.sh -x 2>&1 >s3t-${BUCKET}.log &
done
```

Example *s3t.sh* Log File

This is an example of the *s3t.sh* script log file*:*

```
--> Verify Bucket s3://dt-bucket-1 Exists <--
--> Making Bucket s3://dt-bucket-1 <--
make_bucket: dt-bucket-1
--> Starting Pass 1 <--
--> Removing previous test files <--
--> Creating dt files <--
dt (j:0 t:0): Warning: Top level directory dtfiles-1, will *not* be deleted!
dt (j:1 t:2): End of Write pass 0/1, 72292 blocks, 35.299 Mbytes, 81 records, errors 0/1, elapsed
00m00.46s
dt (j:1 t:5): End of Write pass 0/1, 86525 blocks, 42.249 Mbytes, 89 records, errors 0/1, elapsed
00m00.49s
dt (j:1 t:1): End of Write pass 0/1, 96454 blocks, 47.097 Mbytes, 117 records, errors 0/1,
elapsed 00m00.53s
dt (j:1 t:3): End of Write pass 0/1, 100740 blocks, 49.189 Mbytes, 98 records, errors 0/1,
elapsed 00m00.54s
dt (j:1 t:4): End of Write pass 0/1, 123863 blocks, 60.480 Mbytes, 129 records, errors 0/1,
elapsed 00m00.59s
dt (j:1 t:2): End of Read pass 1/1, 72292 blocks, 35.299 Mbytes, 81 records, errors 0/1, elapsed
00m00.18s
dt (j:1 t:2):
dt (j:1 t:2): Operating System Information:
dt (j:1 t:2):                       Host name: LAPTOP-BJH5MV95 (127.0.1.1)
dt (j:1 t:2):                      Process ID: 2092
dt (j:1 t:2):                  OS information: Linux 5.15.146.1-microsoft-standard-WSL2 #1 SMP Thu
Jan 11 04:09:03 UTC 2024 x86_64
dt (j:1 t:2):
dt (j:1 t:2): File System Information:
dt (j:1 t:2):             Mounted from device: /dev/sdc
dt (j:1 t:2):            Mounted on directory: /
dt (j:1 t:2):                 Filesystem type: ext4
dt (j:1 t:2):              Filesystem options: rw,relatime,discard,errors=remount-ro,data=ordered
dt (j:1 t:2):           Filesystem block size: 4096
dt (j:1 t:2):           Filesystem free space: 251334995968 (239691.730 Mbytes, 234.074 Gbytes,
0.229 Tbytes)
dt (j:1 t:2):          Filesystem total space: 269427478528 (256946.066 Mbytes, 250.924 Gbytes,
0.245 Tbytes)
dt (j:1 t:2):
dt (j:1 t:2): Total Statistics:
dt (j:1 t:2):         Output device/file name: dtfiles-1/j1t2/dt.data-00000010 (device
type=regular)
dt (j:1 t:2):         Type of I/O's performed: sequential (forward, rseed=0x6622ffe2f1928af6)
dt (j:1 t:2):        Job Information Reported: Job 1, Thread 2
dt (j:1 t:2):         Current Thread Reported: 2/5
dt (j:1 t:2):        Data pattern prefix used: dtfiles-1/j1t2/dt.data-00000010@LAPTOP-BJH5MV95@dt-
bucket-1
dt (j:1 t:2):        Data pattern string used: 'IOT Pattern' (blocking is 512 bytes)
dt (j:1 t:2):        Last IOT seed value used: 0x01010101
dt (j:1 t:2):          Block tag verify flags: 0x0fdff7ef
dt (j:1 t:2):              Total records read: 81
dt (j:1 t:2):                Total bytes read: 37013504 (36146.000 Kbytes, 35.299 Mbytes, 0.034
Gbytes)
dt (j:1 t:2):           Total records written: 81
dt (j:1 t:2):             Total bytes written: 37013504 (36146.000 Kbytes, 35.299 Mbytes, 0.034
Gbytes)
dt (j:1 t:2):         Total records processed: 162 with min=512, max=1048576, incr=variable
dt (j:1 t:2):         Total bytes transferred: 74027008 (72292.000 Kbytes, 70.598 Mbytes)
dt (j:1 t:2):          Average transfer rates: 115785156 bytes/sec, 113071.442 Kbytes/sec, 110.421
Mbytes/sec
dt (j:1 t:2):         Number I/O's per second: 253.383
dt (j:1 t:2):          Number seconds per I/O: 0.0039 (3.95ms)
dt (j:1 t:2):          Total passes completed: 1/1
dt (j:1 t:2):           Total files processed: 20/20
dt (j:1 t:2):           Total errors detected: 0/1
dt (j:1 t:2):             Total elapsed time: 00m00.64s
dt (j:1 t:2):              Total system time: 00m00.31s
dt (j:1 t:2):                Total user time: 00m01.17s
dt (j:1 t:2):                  Starting time: Fri Apr 19 13:06:58 2024
dt (j:1 t:2):                    Ending time: Fri Apr 19 13:06:59 2024
dt (j:1 t:2): 
dt (j:1 t:2): Command line to re-read the data:
dt (j:1 t:2):     % /home/robin/dt if=dtfiles-1/j1t2/dt.data-00000010 min=512 max=1048576
incr=vary dsize=512 iotype=sequential iodir=forward limit=37013504 records=7
rseed=0x6622ffe2f1928af6 prefix="dtfiles-1/j1t2/dt.data-00000010@LAPTOP-BJH5MV95@dt-bucket-1"
pattern=iot enable=btags
dt (j:1 t:2): 
dt (j:1 t:2): Command Line:
dt (j:1 t:2): 
dt (j:1 t:2):     % /home/robin/dt of=dtfiles-1/dt.data bs=random min_limit=4k max_limit=10m
incr_limit=vary workload=high_validation threads=5 files=10 dispose=keep iotpass=1 disable=pstats
prefix=%d@%h@dt-bucket-1
dt (j:1 t:2): 
dt (j:1 t:2):         --> Date: December 3rd, 2021, Version: 25.02, Author: Robin T. Miller <--
dt (j:1 t:2): 
    …
```

--> Uploading dt files to S3 server <--

```
Completed 256.0 KiB/~47.1 MiB (2.2 MiB/s) with ~10 file(s) remaining (calculating...)
Completed 512.0 KiB/~47.1 MiB (3.8 MiB/s) with ~10 file(s) remaining (calculating...)
Completed 616.0 KiB/~82.4 MiB (3.7 MiB/s) with ~20 file(s) remaining (calculating...)
upload: dtfiles-1/j1t1/dt.data-00000004 to s3://dt-bucket-1/j1t1/dt.data-00000004
Completed 616.0 KiB/~82.4 MiB (3.7 MiB/s) with ~19 file(s) remaining (calculating...)
Completed 872.0 KiB/~82.4 MiB (4.9 MiB/s) with ~19 file(s) remaining (calculating...)
Completed 1.1 MiB/~82.4 MiB (5.7 MiB/s) with ~19 file(s) remaining (calculating...)
    …
```

--> Downloading S3 dt files <--

```
Completed 256.0 KiB/~26.7 MiB (14.9 MiB/s) with ~7 file(s) remaining (calculating...)
Completed 512.0 KiB/~32.0 MiB (18.4 MiB/s) with ~8 file(s) remaining (calculating...)
Completed 768.0 KiB/~32.0 MiB (26.5 MiB/s) with ~8 file(s) remaining (calculating...)
    …
Completed 3.6 MiB/~50.9 MiB (59.8 MiB/s) with ~11 file(s) remaining (calculating...)
download: s3://dt-bucket-1/j1t1/dt.data-00000004 to s3files-1/j1t1/dt.data-00000004
     …
```

--> Verifying downloaded S3 dt files <--

```
dt (j:1 t:2): 
dt (j:1 t:2): Operating System Information:
dt (j:1 t:2):                      Host name: LAPTOP-BJH5MV95 (127.0.1.1)
dt (j:1 t:2):                     Process ID: 2265
dt (j:1 t:2):                 OS information: Linux 5.15.146.1-microsoft-standard-WSL2 #1 SMP Thu
Jan 11 04:09:03 UTC 2024 x86_64
dt (j:1 t:2): 
dt (j:1 t:2): File System Information:
dt (j:1 t:2):            Mounted from device: /dev/sdc
dt (j:1 t:2):           Mounted on directory: /
dt (j:1 t:2):                Filesystem type: ext4
dt (j:1 t:2):             Filesystem options: rw,relatime,discard,errors=remount-ro,data=ordered
dt (j:1 t:2):          Filesystem block size: 4096
dt (j:1 t:2):          Filesystem free space: 249866145792 (238290.926 Mbytes, 232.706 Gbytes,
0.227 Tbytes)
dt (j:1 t:2):         Filesystem total space: 269427478528 (256946.066 Mbytes, 250.924 Gbytes,
0.245 Tbytes)
dt (j:1 t:2): 
dt (j:1 t:2): Total Statistics:
dt (j:1 t:2):         Input device/file name: s3files-1/j1t2/dt.data-00000010 (device
type=regular)
dt (j:1 t:2):        Type of I/O's performed: sequential (forward, rseed=0x6622fffcfbeb1d57)
dt (j:1 t:2):       Job Information Reported: Job 1, Thread 2
dt (j:1 t:2):        Current Thread Reported: 2/5
dt (j:1 t:2):       Data pattern prefix used: s3files-1/j1t2/dt.data-00000010@LAPTOP-BJH5MV95@dt-
bucket-1
dt (j:1 t:2):       Data pattern string used: 'IOT Pattern' (blocking is 512 bytes)
dt (j:1 t:2):       Last IOT seed value used: 0x01010101
dt (j:1 t:2):         Block tag verify flags: 0x0f8841e7
dt (j:1 t:2):             Total records read: 78
dt (j:1 t:2):              Total bytes read: 37013504 (36146.000 Kbytes, 35.299 Mbytes, 0.034
Gbytes)
dt (j:1 t:2):          Total records written: 0
dt (j:1 t:2):            Total bytes written: 0 (0.000 Kbytes, 0.000 Mbytes, 0.000 Gbytes)
dt (j:1 t:2):        Total records processed: 78 with min=512, max=1048576, incr=variable
dt (j:1 t:2):        Total bytes transferred: 37013504 (36146.000 Kbytes, 35.299 Mbytes)
dt (j:1 t:2):         Average transfer rates: 118805903 bytes/sec, 116021.390 Kbytes/sec, 113.302
Mbytes/sec
dt (j:1 t:2):        Number I/O's per second: 250.364
dt (j:1 t:2):         Number seconds per I/O: 0.0040 (3.99ms)
dt (j:1 t:2):         Total passes completed: 1/1
dt (j:1 t:2):          Total files processed: 10/10
dt (j:1 t:2):          Total errors detected: 0/1
dt (j:1 t:2):             Total elapsed time: 00m00.31s
dt (j:1 t:2):              Total system time: 00m00.08s
dt (j:1 t:2):                Total user time: 00m01.17s
dt (j:1 t:2):                  Starting time: Fri Apr 19 13:07:08 2024
dt (j:1 t:2):                    Ending time: Fri Apr 19 13:07:08 2024
dt (j:1 t:2):
dt (j:1 t:2): Command Line:
dt (j:1 t:2):
dt (j:1 t:2):     % /home/robin/dt if=s3files-1/dt.data bs=random workload=high_validation
vflags=~inode threads=5 files=10 iotpass=1 disable=verbose prefix=%d@%h@dt-bucket-1
dt (j:1 t:2):
dt (j:1 t:2):         --> Date: December 3rd, 2021, Version: 25.02, Author: Robin T. Miller <--
dt (j:1 t:2):
    …
```

--> Removing S3 dt files <--

```
delete: s3://dt-bucket-1/j1t1/dt.data-00000001
delete: s3://dt-bucket-1/j1t1/dt.data-00000002
delete: s3://dt-bucket-1/j1t1/dt.data-00000003
delete: s3://dt-bucket-1/j1t1/dt.data-00000004
delete: s3://dt-bucket-1/j1t1/dt.data-00000005
delete: s3://dt-bucket-1/j1t1/dt.data-00000006
delete: s3://dt-bucket-1/j1t1/dt.data-00000007
delete: s3://dt-bucket-1/j1t1/dt.data-00000008
delete: s3://dt-bucket-1/j1t2/dt.data-00000001
delete: s3://dt-bucket-1/j1t1/dt.data-00000010
```