EMC²
where information lives

iorate

Open Systems I/O Driver and Measurement Tool

User Guide

A copy of this *iorate Open Systems I/O Driver and Measurement Tool* User Guide is available at: **`ftp://ftp.emc.com/pub/elab/iorate/`**.

Table of Contents

Introduction to iorate

iorate, an EMC®-created tool, is designed for Open Systems customer storage benchmarking. The tool can perform a series of I/O tests on devices under various loads, and works between a host and a disk subsystem. The I/O driven can be against files on file systems or against raw devices. There is no EMC or Symmetrix™-specific code; it is all generic to storage devices.

Each I/O test is defined using a combination of configuration files. These include the devices against which to test, the type of I/O patterns, and the mix of patterns for the tests. The configuration files can be default patterns or user-designed. The test runs against all of the devices simultaneously. There can be multiple tests defined, each one running after the previous one has completed. iorate generates a detailed log file of the tests being run so that users can reproduce a test. In addition, iorate generates a performance file which contains detailed statistics on each test run on each device. Users can define multiple tests, and each one will run in succession after the completion of the previous test.

> **WARNING:** The iorate tool can be destructive to the data on all files/devices used for testing. If there is data in the files/devices, include these files/devices in the device file only if you have taken extreme care to ensure that all test patterns are read-only.
>
> Because of this danger, the default test file in the distribution uses read-only patterns. However, you should take great care whenever any entry in the devices file has data that you value. Failure to do so can result not only in lost data, but can cause damage to the root file system (writes of garbage data) if those devices are included in the tests. There is no attempt made to check for validity of devices under test or for other users of any device listed; the tests are simply executed.

This guide has three main sections; 1. Preparations for Testing, 2. Running the Tests, and 3. Interpreting Test Results. Please reference each section for specific information on each topic.

1. Preparations for Testing

A test is a group of I/O patterns that are applied to a device.

1. Determine your goals for the test. Once you know the goals for the test, you can then decide what devices you test against as well as the types of tests to run against those devices or files/filesystems.
2. Determine what you want to test: which devices or filesystems. The devices or filesystems will become our Devices File (devices.ior) as outlined in the Devices File section.
3. Determine what type of I/O (Random or Sequential, Read or Write) to run. This is the information which forms our Patterns File (patterns.ior) as outlined in. Patterns File.
4. Determine the sequence of tests that execute the patterns. Sequencing of the tests is stored in our Tests File (tests.ior) as outlined in the Tests File section.
5. Once you determine what you want to test, decide the length of time for the test and determine if you require a portion of the time for the test to get into a steady state or to ramp-up before you start measuring statistics.

6. Use a text editor (vi or emacs, for example) to edit the configuration files to match your requirements as defined in the Configuration Files section.
7. You can run iorate using either the default filenames or the ones you edited and renamed. The 2. Running the Tests section discusses this procedure. iorate will generate a .log and a .perf file which we'll look at in the 3. Interpreting Test Results section.

Configuration Files

Use the three default configurations contained in the distribution, or edit and customize them.

The three required configuration files are:

- Devices.ior: The Device_file lists the devices to test.
- Patterns.ior: The Pattern_file lists the I/O types that can be run against a device.
- Tests.ior: The Tests_file lists the tests to run.

In general, anything following a hash mark (#) is a comment to the end of the physical line (marked by a carriage return). Each logical line describes a single item (device, pattern, test, or IOPS rating as outlined below) and ends in a semicolon (;). Logical lines (single items) can span many input lines.

Each item will have an item type identifier (device, pattern, test, or target) and then information about that item. The details of the various options for each item type are detailed throughout this section.

When providing sizes, a number can be followed by a sizing factor (KB, MB, GB, and TB). Each step in the sizing factor will increase the size by a multiplier of 1024 (not 1000). When providing time values, you must follow a number by a time type (sec, min, and hour) which will scale the number to seconds automatically within the tests.

Devices File

The devices file is used to define the devices or files that I/O can be performed against. The default filename without specifying the –f modifier is devices.ior.

Each device entry describes a single device, such as:

Example:

```
Device 1 = "/mnt/datafile.tst" capacity 3GB;
```

You may use any valid file name, whether a raw device file, or the name of a filesystem file that exists and has the stated size. Device numbers must be unique, but are also optional. iorate will assign a device number sequentially automatically if you have not designated one. You can view these numbers in the log file output (iorate.log).

Note: The term *device* is used in this document to mean a raw device or a filename.

Capacity refers to the requested amount of space in the device for testing; the actual size of the device can be much larger. The capacity of the device is the requested capacity starting at the offset point. The offset is used to position the test start position past device header data such as disk signatures, master boot records, and similar information. Most device header information is located within the first few KB of the device and we don't want access to that information. For example, I/O can be targeted at the last 2 GB of a 36 GB device using offset 34 GB capacity 2 GB.

There is a minimum default offset of 8 KB to prevent overwriting of device header information. You can set alternate values with *offset* <size>.

iorate defaults to a single instance of a device for testing. For testing multiple threads, you can include multiple instances of the device or add the *count* <integer> modifier. This will create <integer> threads of I/O to the device. In the following example, there will be eight threads assigned to the device, starting at an offset of 64 KB.

Example:

```
Device = "/dev/rdsk/c1t0d0s2" capacity 16.9GB count 8 offset
64KB;
```

The minimum default block size is 512 bytes for most operating systems (it is 1024 bytes on HP-UX). This is due to the format of fixed block address (FBA) implementations of most operating systems. This block size is the minimum I/O size for the device, and all I/O will be aligned to this size, as well as being a multiple of it. The patterns in use for any test must be a multiple of this size.

The device can be designated as being read-only by specifying *read only* on the device line. This will cause iorate to open the device for reads but not writes, and any tests attempting to do writes will cause iorate to fail when it attempts to write to the device. Unless the device is set to read-only, iorate will attempt to open the device for reads and writes - even if none of the active patterns in the tests will do writes. Specifying *read only* is useful if you are going to test on production data, or if the files/devices you want to test on are not writable by the user who will be running the tests.

Note: The failure `ERROR: BAILING out of test due to errors` will occur when iorate attempts to write to a device to which it doesn't have write access. Any read tests to the device prior to the write test will succeed.

Example:

```
Device = "/dev/rdsk/c1t0d0s2" capacity 16.9GB count 1 offset 8KB
read only;
```

In addition, there is an option to lock the file by specifying *lock*. This will use the *fcntl()* system call to lock the area of the device specified by the offset and capacity. You add *lock* to the device line and the region under test (from *start* and going for *size*) will be locked before any I/O is performed (and unlocked when completed). This is useful to verify that you do not have overlapping areas of a file. Note that these are exclusive (for writes), non-blocking locks: if there can be no locking, the entire test will be aborted. There can be multiple read locks on the same area of a file, so setting the devices to read only will enable them to be shared even if locking is requested. If the device is not set to read-only, iorate will attempt to acquire a write lock (exclusive) on the defined region of the file, even if none of the active patterns in the tests will do writes.

Example:

```
Device 1 = "/dev/rdsk/c1t0d0s2" offset 64KB capacity 8.40GB read
only lock ;
```

Table 1: Sample Devices.ior from Distribution

```
# File system example
Device = "/mnt/null/trash1.dat"  capacity 4.1GB;

# UNIX raw device example
Device = "/dev/rdsk/c901t0d0s2"  capacity 4.1GB;

# Windows raw device example
Device = "\\.\PHYSICALDRIVE200"  capacity 4.1GB;
```

The default file entry in Table 1 shows three different types of devices: a file in a file system to be tested, a UNIX® raw device, and a Windows™ raw device.

Note: This is the default file that comes with the distribution. The file gives syntax examples, and users should edit the file before running any tests. Failure to do so can cause data corruption in the event you have devices that match the entries in the default devices.ior file.

Note: Very important. The boot volume may be listed (as shown below `/dev/rdsk/c1t0d0s2`), so you will have to delete that line (preferred) or comment it out with a "#" sign at the beginning of the line.

Below is an excerpt from a gen_dev run under Solaris™ x86. The gen_dev script enables you to generate a devices file automatically. gen_dev is detailed in the Additional Utility Scripts section of the Appendix.

Example:

```
#
#  Automatically generated list of raw devices from syminq
#
#  Remove any devices that contain file systems or raw
#  device data or DATA LOSS will result.  Check for
#  alternate path devices such as those from Veritas DMP
#  or PowerPath.  Also be sure to fix/remove any devices
#  whose name may not be complete (syminq uses '**' at
#  the end for device names that are too long.)
#
REMOVE THIS LINE ONCE DEVICES ARE CHECKED
Device = "/dev/rdsk/c1t0d0s2" capacity 16.9GB count 1 offset 8KB;
Device = "/dev/rdsk/c3t0d0s2" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rdsk/c3t0d1s2" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rdsk/c3t0d2s2" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rdsk/c3t0d3s2" capacity 8.4GB count 1 offset 8KB;
```

If you have Solutions Enabler installed, you can review and select which devices you want to include in your testing by running "`syminq –pdevfile`" and matching the output to the list of directors and ports against which you want to test. Here is a sample output:

```
> syminq –pdevfile
# Symm_id        pdev             dev dir dir_port
000187400160 /dev/rdsk/c3t0d0s2 0A0 13C  0
000187400160 /dev/rdsk/c3t0d1s2 0A1 13C  0
000187400160 /dev/rdsk/c3t0d2s2 0A2 13C  0
000187400160 /dev/rdsk/c3t0d3s2 0A3 13C  0
000187400160 /dev/rdsk/c3t0d4s2 0A4 13C  0
```

This tells us the Symmetrix devices, 0A0 through 0A4 exist on logical /dev/rdsk/c3t0d0 through /dev/rdsk/c3t0d4 and are on director 13C, Port 0.

Patterns File

The *patterns file* is used to define the different I/O types that can be performed on a device. The default filename without specifying the –p modifier is patterns.ior.

You can choose to use the default I/O patterns included in the distribution or edit and create your own. The patterns file describes each block size and type of I/O that you will be running in your test. These patterns will consist of either read or write and either random or sequential. These are the different I/O types that can be performed on a device.

You must specify either read or write for the pattern. You must also specify random or sequential. Setting I/O random will change the pattern to random address selections. To select a pattern that runs a specified number of I/Os in a sequence before moving to another random location, use *max sequential* <integer> to limit the number of sequential I/Os to be done in a row. Setting *max sequential* <integer> to 10 will always do 10 sequential I/Os before selecting a new location. To mix the I/O types, select write <integer>% or *read* <integer>% to have the pattern do a mix of reads and writes.

Example:

Each pattern entry describes a single pattern, such as:

Pattern 1 = "2k Seq Read" io size 2KB max seq 3 sequential read;

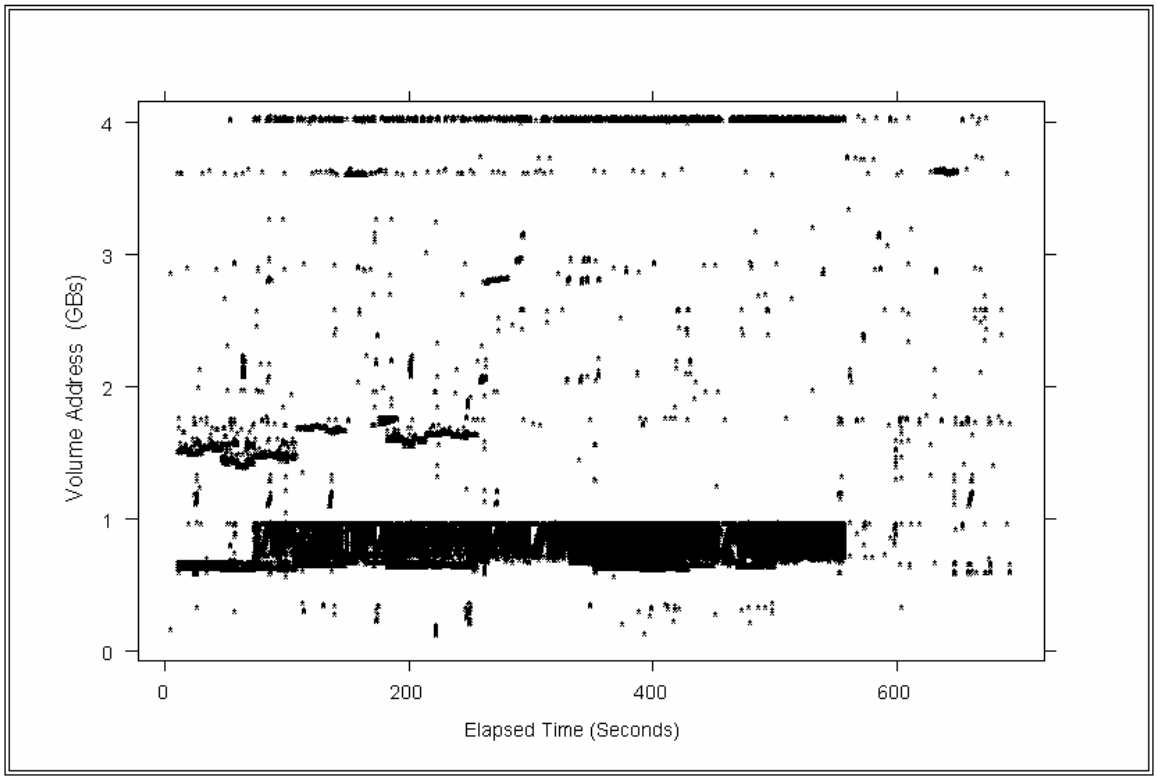
These pattern numbers will be used when you create the tests file described in the Tests File section. The patterns do not need to be numbered consecutively, but you must not duplicate them.

Most applications access the same sections of a device frequently. To model a real application more closely, you may need to generate I/Os to the same address (es) to simulate cache hits. iorate provides users the capability to perform this type of I/O to the devices. The *reuse* option specifies how much of your I/O will be to recently- addressed locations. Setting *reuse* <integer>% will cause that percentage of I/Os to be read or written to the same address as another recent I/O. For example, setting reuse 50% will cause half of your I/Os to be to addresses that were recently accessed. The *history* <integer> option will define the number of I/Os you wish to track.

Example:

Pattern 20 = "2k Mixed random 50% Hit" io size 2KB random read 100% history 250 reuse 50%;

Figure 1 – Locality of Reference



In analyzing real life applications, it was discovered that they access devices in a non-random way. This is demonstrated in Figure 1 – Locality of Reference. You can create these 'locality of reference' zones for testing.

Figure 1 shows random data accesses to a particular data volume. You can see that there is higher activity at a portion of the volume around 500 MB - 1 GB. You can simulate testing this area by selecting a locality of reference zone. By setting *zone* <size> as a fixed size, or *zone* <integer>% as a percentage of the pattern test area, iorate will create *count* <integer> localities against which to run I/Os. To model multiple, moving hot spots, specify *limit* <integer> to determine how many I/Os to read or write to a given locality before moving to a new location.

Example:

Pattern 22 = "4k rand read 4 zones" io size 4KB read 100% random **zone 5% count 4 limit 30** ;

Tests File

The tests file is used to define the different I/O patterns that can be performed on a device or group of devices. The default filename without specifying the –t modifier is tests.ior. A test is a group of I/O patterns that are applied to a device.

Each test uses one or more patterns. (Refer to Patterns File section) The patterns are specified as a percentage of the I/Os to be performed. The tests specify how long they

should run, where they start in the file/device, and the size of the device that they should use for testing. They can also specify how much of the test period should be ignored for performance characteristic reasons to enable time for the test to achieve a steady state, which can be useful when testing disk arrays with large amounts of cache.

Each test entry describes a single test, such as:

Example:

Test 1 = "Mixed 2K Seq" for 120 sec ignore 20 sec 50% pat 1, 50% pat 2;

Test numbers must be unique, but are also optional. If you do not provide a test number, iorate will assign one in order automatically, and provide a record in the log file.

The duration of the test is set with *for* <time>. <time> can be specified in units of seconds, minutes or hours.

The I/O *size* <size> parameter specifies the size of all I/Os that will be driven from this pattern. To mix I/O sizes, users may configure tests that use multiple patterns.

To ensure that iorate measures the steady-state numbers and does not measure the performance during any start-up period, set *ignore* <time> which causes the performance statistics to be reset that far into the test time. In the event that one test leaves work in the test storage, such as writes sitting in the cache of a disk array, *pause* <time> will introduce that length of delay before activating the test.

Users may specify the area against which to run the test *from* <size>, or *from* <integer>% to set the starting location, and *size* <size> or size <integer>% to set the size against which to test (starting at the *from* location.) Sizing that is based on percentages is based on the capacity of each device being tested – so, if there are devices with different capacities, the tests will resize themselves for each one.

Example:

Test 2 = " Monday test" pause **0 sec for 2 min ignore 1 min from 8KB size 1GB** 100% pat 1 ;

The *from 8 KB* determines the offset at which to start the testing, followed by the size of testing. In the example for Test 1 above, we skip the first 8 KB and test the next 1 GB. To limit the level of I/O that this test generates, set <integer> *iops*. This sets an I/Os per second target for the entire test. Since there are separate processes run for each defined active device, the processes will target their shares independently. This means that if there are 10 active devices, and 100 Iops specified, the program will attempt to drive 10 Iops to each device. In the event that one device cannot do its share, the other devices will not be tasked with additional work, as the test process working on each device is not aware of the others. If "*0 iops*" is used, it means the test will "*run at maximum*".

Example:

Test 2 = " Monday test" pause 0 sec for 2 min ignore 1 min from 8KB size 1GB **500 iops** 100% pat 1 ;

iorate starts one process per device (more if count <integer> is specified in the Devices File section). Each process attempts to perform the mixture of I/O to its respective device that was specified in the tests.ior file. The overall workload is divided among the total number of processes.

2. Running the Tests

Run Time Requirements

iorate requires that all three of the required configuration files described in the Configuration Files section be properly formatted. Any errors in the files will be reported as the program starts, and may abort the testing. For a complete list of command line options, please refer to the Command Line Syntax (with optional parameters in brackets) and Optional Command Line Parameters sections in the Appendix

Initiate the testing by invoking iorate with optional configuration file names and parameters. With no configuration files specified on the command line, iorate defaults to the file names listed in the Configuration Files section with iorate.perf and iorate.log as default output files (unless specified with –o option).
The tests will run in consecutive order and write their results to the .log and .perf output files.

Example Run Parameters

- ./iorate –p patterns.ior –f devices.ior –t tests.ior
  This invokes iorate with the default configuration file names.
- ./iorate -p customerX_pat.ior -f customerXdevs4proc.ior -t customerX_tests.ior -o customerX4proc
  This invokes iorate with customized pattern, device and test configuration file names. It also outputs to customerX4proc.log and customerX4proc.perf.

3. Interpreting Test Results

Log File

iorate generates a log file of the test being run. The default filename without specifying the –o modifier is iorate.log.

The log file contains a copy of the configuration files being used (device_file, pattern_file and test_file). These will not have any comments, and will be printed directly from the information that was used to drive the actual tests. These copies, if split out into the separate input files, will reproduce the exact same test. They are included in the log to show exactly what was run, and to enable that test to be reproduced at a later time. The log also records the starting and ending times of each test by device. Any errors encountered are also included in the log.

The log file is usually named iorate.log. If the ***-o*** <output_base> option is used, log file will be output_base.log. If large numbers of test passes will be run, you should build sub-directories for each group of tests. The output_base parameter can include sub-directory names as well, but those sub-directories must exist before iorate begins (iorate will not create sub-directories.)

If an iops_file was used as detailed in the Optional Files section of the Appendix, or if the target rate was scaled using a target_rate (see Optional Command Line Parameters section), then the iops target for each test in the test_file section of the log will have a value reflecting these target rate adjustments. Splitting these sections out to run a new test will run the exact same test again, at the adjusted target rate, without the need for target

iops or rate adjustments. If the adjustments (File and/or target_rate percentage) are applied again, the tests may be altered.

Example:

```
Starting IORATE (./iorate) version 2.6
- - - - - - Log of Devices Data from 'rwm4kdev.ior'- - - - - -
Device 1 = "/dev/rhdisk975"  offset 8KB capacity 8.40GB block
size 512B ;
Device 2 = "/dev/rhdisk976"  offset 8KB capacity 8.40GB block
size 512B ;
Device 3 = "/dev/rhdisk977"  offset 8KB capacity 8.40GB block
size 512B ;
- - - - - - End of Devices Data - - - - - -


- - - - - - Log of Pattern Data from 'rwm4kpat.ior'- - - - - -


Pattern 1 = "4k Seq Read"  io size 4KB  read 100%  sequential ;
Pattern 2 = "4k Random Read"  io size 4KB  read 100%  random ;
Pattern 3 = "4k Seq Write"  io size 4KB  read 0%  sequential ;
Pattern 4 = "4k Random Write"  io size 4KB  read 0%  random ;
- - - - - - End of Pattern Data - - - - - -
- - - - - - Log of Tests Data from 'rwm4ktests.ior'- - - - - -
Test 1 = "4k RWM Test 64 devices 16 devices per port" for 20 sec
     0 iops
     100% pat 4, ;
- - - - - - End of Tests Data - - - - - -
     Beginning test 1 '4k RWM Test 64 devices 16 devices per
port':
               Ignore: 0 sec
               Duration: 20 sec
               I/Os per Second: Not limited
          100% pattern 4 '4k Random Write'
Test 1 starting, file '/dev/rhdisk975', device 1 copy 1, at Tue
Nov 09 2004 15:56:28
Test 1 starting, file '/dev/rhdisk976', device 2 copy 1, at Tue
Nov 09 2004 15:56:28
Test 1 starting, file '/dev/rhdisk977', device 3 copy 1, at Tue
Nov 09 2004 15:56:28
Test 1 finished, file '/dev/rhdisk975', device 1 copy 1, at Tue
Nov 09 2004 15:56:49
Test 1 finished, file '/dev/rhdisk976', device 2 copy 1, at Tue
Nov 09 2004 15:56:49
```

```
Test 1 finished, file '/dev/rhdisk977', device 3 copy 1, at Tue
Nov 09 2004 15:56:49
```

Performance File

In addition to the log file, iorate generates a performance file for the test group. The default filename without specifying the –o modifier is iorate.perf.

The performance file contains detailed statistics on each test run on each device. Each line in the file represents the data for a single test on a single device. The first line of the file is a header that describes the contents of each test line. The file uses tab separated values that are easy to parse with scripts or import into spreadsheet programs (though the alignments are a challenge if read with a standard editor or simply printed). The fields are:

test_number -- Number of the test being run

test_name -- Name of the test being run

device_num -- Number of the device for this line

device_name -- Name of the device for this line

total_sec -- Total seconds the test was run

measured_sec -- Time over which results are measured

Reads -- Number of reads from this device

KB_read -- KB of data reads from the device

read_resp -- Avg. response time for all device reads

Writes -- Number of writes to this device

KB_written -- KB of data written to the device

write_resp -- Avg. response time for all device writes

Total_I/Os -- Number of I/Os to this device

total_KB -- KB of data transferred to the device

total_resp -- Avg. response time for all device I/Os

I/Os_per_sec -- Avg. number of device I/Os per second

KB_per_sec -- Avg. KB of data transferred per second

dev_copy – Number of threads to the device

Each line is a set of performance numbers for a single device. To get the total for a given test, summarize the numbers for each of the devices for each test. The gen_sums AWK program detailed in the Additional Utility Scripts section will generate such test summaries, which are often more useful for comparisons. If the pair of output files (performance and log) is available from any test, users may examine the numbers and reproduce the test.

Example:

```
test_number      test_name  device_num device_name      total_sec
      measured_sec     reads KB_read    read_resp  writes
      KB_written write_resp total_I/Os total_KB   total_resp
      I/Os_per_sec     KB_per_sec dev_copy
1     4k RWM Test      1     /dev/rhdisk975   20    20    0     0
      0.000 6433  25732 3.014 6433  25732 3.014 321   1286  1
1     4k RWM Test      2     /dev/rhdisk976   20    20    0     0
      0.000 6424  25696 3.018 6424  25696 3.018 321   1284  1
1     4k RWM Test      3     /dev/rhdisk977   20    20    0     0
      0.000 6138  24552 3.120 6138  24552 3.120 306   1227  1
```

As you can see in this example, the output is very congested due to multiple fields overrunning a standard screen output. You can load the output into a spreadsheet program such as Excel™ for easier manipulation.

Here are the various fields for the three devices listed above:

test_number -- **1**

test_name -- **4k RWM Test**

device_num – **1, 2 and 3**

device_name -- **/dev/rhdisk975, /dev/rhdisk976 and /dev/rhdisk977**

total_sec -- **20**

measured_sec -- **20**

Reads -- **0**

KB_read -- **0**

read_resp – **0.000**

Writes – **6433, 6424 and 6138**

KB_written – **25732, 25696 and 24552**

write_resp -- **3.014, 3.018 and 3.120**

Total_I/Os – In this case, since there are no reads, this matches Writes

total_KB -- In this case, since there are no reads, this matches KB_written

total_resp -- In this case, since there are no reads, this matches write_resp

I/Os_per_sec – **321, 321 and 306**

KB_per_sec -- **1286, 1284 and 1227**

dev_copy – **1**

The performance file is usually named iorate.perf. If you use the ***-o*** <output_base> option, the performance data will be output_base.perf. If you run large numbers of test passes, EMC recommends that you build sub-directories for each group of tests. The output_base parameter can include sub-directory names as well, but those sub-directories must exist before iorate begins (iorate will not create sub-directories.)

You can process the .perf file using the utilities detailed in the Additional Utility Scripts section. If you process the data using gen_sums or gen_totals, you will get an .xls file that you can import into Excel or parse for your own purposes. This is useful for comparing successive runs or varying configurations.

Appendix

Optional Files

iops_File

If there is an iops_file on the command line or specified through the environment variable, the iops limit (throttle) will be set for that run of the test. The filename is required when specifying the –i modifier.

If there is no iops limit specified for a test, then I/O will be generated as fast as possible on every device. When you specify an iops_file, each entry sets a new iops limit for a given test, such as:

Example:

```
Target test 5 at 2000 iops;
```

This is most useful for times when you want to run a given set of tests on different configurations, and the program does not know ahead of time what the upper limit of each test will be (maximum test rate). From there, it may be interesting to run the same tests at some percentage of the maximum rate.

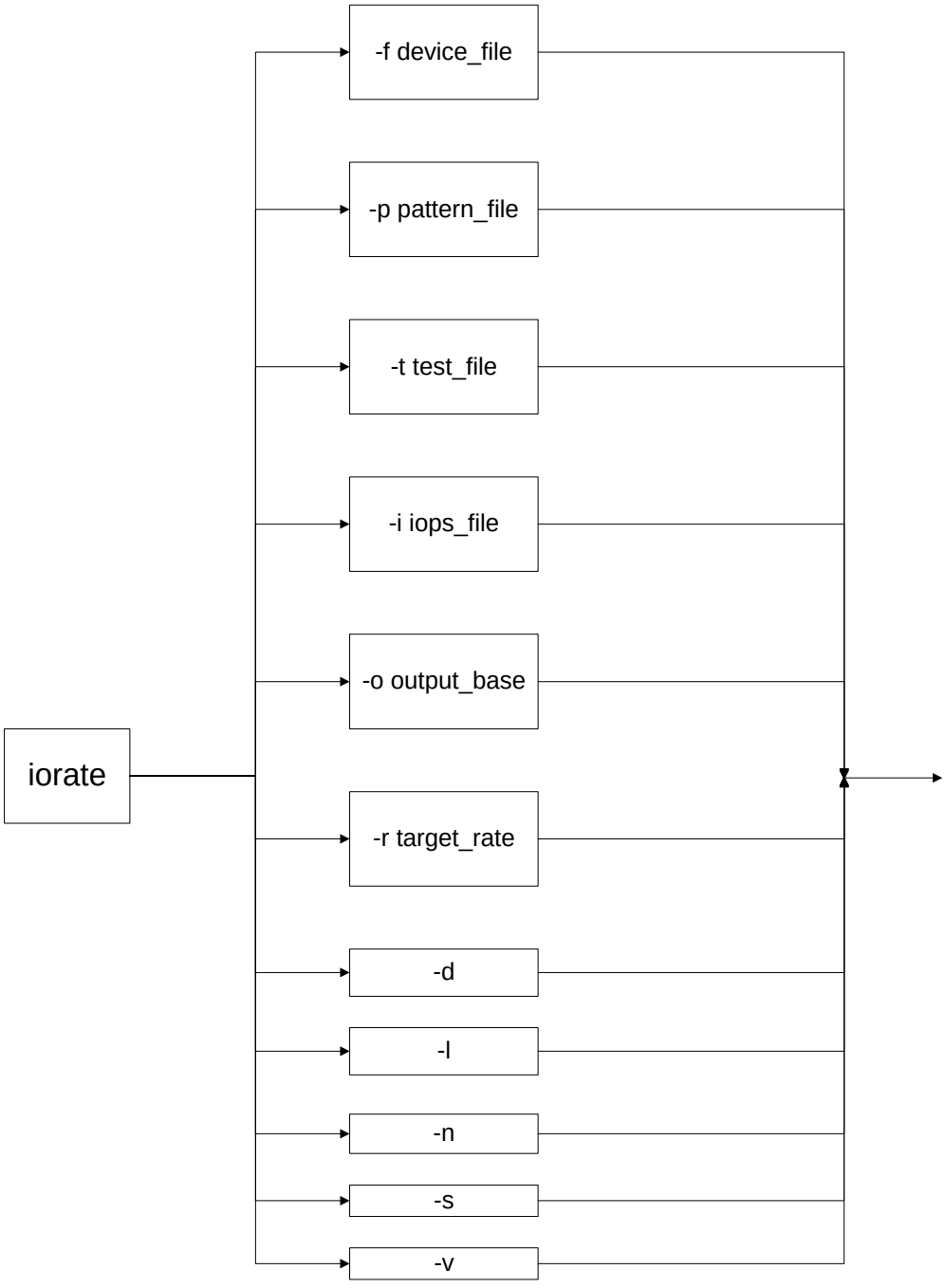
If you have run a full speed pass of a test already, then the gen_iops AWK program detailed in the Additional Utility Scripts section, will generate an iops_file that matches the performance you see during that run. Together with a new target_rate, you can test a pattern for response times at various percentages of the maximum attained throughput for each test.

Example:

```
Target test 1 at 44657 iops;
Target test 2 at 27107 iops;
Target test 3 at 5723 iops;
```

Command Line Syntax (with optional parameters in brackets)

```
iorate [ -d ] [ -l ] [ -n ] [ -s ] [ -v ] [ -r target_rate ]
      [ -f device_file ] [ -p pattern_file ] [ -t test_file ]
      [ -i iops_file ] [ -o output_base ]
```



Optional Command Line Parameters

The ***-l*** (limits) option causes iorate to display file limits and exit (no testing). This is used in the makefile to verify support for large file sizes.

The ***-n*** (no testing) option turns off test execution. The program will still do all setup checks, including a seek to test the size of each file. The program will still create a child process to do the testing. However, the child will exit quickly.

The ***-s*** (silent) option stops all but the error and warning output to standard out. The other information normally reported, such as test configurations and progress, is still recorded in the log file for later reference.

The ***-d*** (debug) option turns on a number of debugging outputs. This will provide a large number of messages that may not be understandable to someone not reading the program source code.

The ***-v*** (verbose) option turns on verbose mode. Since this will generate log information for every I/O of every device, you should not use it in a lengthy test. It will have a dramatic effect on the I/O rates of many tests; therefore you should not turn it on while generating real test numbers. Note that multiple processes (one per device) will be writing to the same log file, so the output lines may get a bit messy with more than one active device. The log file will be very busy, and may grow to be quite large.

The ***-r*** (target_rate) tells iorate to run at a reduced target rate. This will cause each test to run at this percentage of the maximum iops that you entered for that test. (This has no effect if an iops has not been specified for that test.)

You may specify the target (base) iops rate for each test in the test_file, or in the iops_file. Valid target rates are from 0 (no target rate used - leave iops values alone - to 100% [full test].)

Environment Variables

The command line file names can be provided for in the environment, with:

IOR_DEV_FILE

The name of the device_file to use.

IOR_PAT_FILE

The name of the pattern_file to use.

IOR_TEST_FILE

The name of the test_file to use.

IOR_IOPS_FILE

The name of the iops_file to use.

IOR_OUTPUT_BASE

The base name of the output files.

Additional Utility Scripts

To run the additional Utility Scripts, you must install AWK on your system to process the output files.

gen_devs utility script

A utility script, gen_devs, exists in the distribution directory, which enables you to generate a devices file. You will need to install Solutions Enabler and awk to run gen_devs If you run gen_devs, it accesses the gen_dev.awk script which runs a syminq and formats the output in a format that iorate can read. The script outputs to standard output (the screen only) so you will need to redirect to a filename if you intend to use it in your testing.

Example:
```
gen_devs > customer_devs.ior
or
gen_devs | tee customer_devs.ior
```

Note that you will have to remove the line "`REMOVE THIS LINE ONCE DEVICES ARE CHECKED`" before you can run any tests. Failure to delete the line will disable any testing until the line is removed.

gen_sums utility script

A utility script, gen_sums exists in the distribution directory. When you run this script, you will get a summary of each test. The script takes all the filename.perf files and summarizes all line items into a single line summary.

As you can see in the following example, the output is very congested due to multiple fields overrunning a standard screen output. You can load the output into a spreadsheet program such as Excel for easier manipulation.

Example:

```
# gen_sums
starting on file rwm4k60sec.perf
    ---> generated rwm4k60sec.xls
Output of rwm4k60sec.xls:
# cat rwm4k60sec.xls
test_number    test_name       total_sec     measured_sec   reads
KB_read    read_resp      writes  KB_written    write_resp
total_I/Os     total_KB
        total_resp     I/Os_per_sec   KB_per_sec
1          4k RWM Test       60        15          0     0
      0.0        262492  1049968 3.65       262492     1049968
      3.65       17499      69997
```

gen_totals utility script

A utility script, gen_totals exists in the distribution directory. When you run this script, it executes gen_sums on the output files in the data directory and you will get a summary of the last three tests run.

As you can see in the following example, the output is very congested due to multiple fields overrunning a standard screen output. You can load the output into a spreadsheet program such as Excel for easier manipulation.

Example:

```
      # gen_totals
      starting on file rwm4ktests_4pass.perf
          ---> generated rwm4ktests_4pass.xls
Output of rwm4ktests_4pass.xls:


      # cat rwm4ktests_4pass.xls
      test_number     test_name       total_sec
measured_sec    reads   KB_read
      read_resp       writes  KB_written      write_resp
total_I/Os      total_KB
              total_resp      I/Os_per_sec    KB_per_sec
      1       4k RWM Test Pass 1      15      5       0       0
0.0     26640
      106560  2.99    26640   106560  2.99    5328    21312
      2       4k RWM Test Pass 2      30      20      0       0
0.0     101552
      406208  3.14    101552  406208  3.14    5077    20310
      3       4k RWM Test Pass 3      45      35      0       0
0.0     105146
      420584  5.31    105146  420584  5.31    3004    12016
```

Installation/Compilation

Locating iorate

iorate is available at: **ftp://ftp.emc.com/pub/elab/iorate/**.

The site includes binary versions for some platforms as well as the source code. iorate is available for most platforms. Since most platforms offer upward/forward compatibility, you will not need to recompile unless the newer version of the OS has additional functions/features (such as large device or 64 bit compliance.) For this reason, the source package is available.

Loading Source Files

Downloading source files depends on which operating system you will be using for the test. Users should determine if a binary/executable for your platform exists. If not, you must compile the source.. If the target operating system is not available, users must use the C compiler and associated tools as listed in the Compiler Requirements section.

Compiler Requirements

If an executable isn't compiled and available for the target system, iorate requires a C compiler, and yacc or bison. If an executable is available, the executable can be run without recompiling.

If there is a need for compilation, you may need to define/edit makefiles to specify compiler options and to help some platforms work with files greater than two GB. New defines are included for AIX and HP-UX, but you must manually adjust the makefile for these environments to use 64-bit file access.

Platform specific issues

Solaris x86™

You will need to change the slice designation from s2 to p0. If you don't, you will get the following error:

```
/dev/rdsk/c4t0d0s2: I/O error
./iorate: ERROR: Attempt to open file 33 '/dev/rdsk/c4t0d0s2' failed
```

PowerPath™

Here is an example of the gen_devs output for an AIX host with PowerPath installed:

```
#
REMOVE THIS LINE ONCE DEVICES ARE CHECKED
Device = "/dev/rhdiskpower16" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdiskpower17" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdiskpower106" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdiskpower107" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdisk474" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdisk475" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdisk476" capacity 8.4GB count 1 offset 8KB;
Device = "/dev/rhdisk477" capacity 8.4GB count 1 offset 8KB;
```

Note that there are both rhdisk and rhdiskpower devices. To see the underlying devices behind an rhdiskpower device, you can use the command "`powermt display dev=<device>`".

Example:

```
>powermt display dev=hdiskpower198
Pseudo name=hdiskpower198
Symmetrix ID=000187400160
Logical device ID=023F
state=alive; policy=SymmOpt; priority=0; queued-IOs=0
==============================================================================
------------- Host ------------   - Stor -   -- I/O Path -  -- Stats ---
### HW Path                I/O Paths    Interf.   Mode    State  Q-IOs
Errors
==============================================================================
  5 fscsi5              hdisk535   FA 14dB   active  alive      0      0
  1 fscsi1              hdisk726   FA  3dB   active  alive      0      0
  0 fscsi0              hdisk788   FA  3cB   active  alive      0      0
  2 fscsi2              hdisk850   FA  3aB   active  alive      0      0
  3 fscsi3              hdisk912   FA  3bB   active  alive      0      0
  4 fscsi4              hdisk974   FA 13dB   active  alive      0      0
```

The example shows device hdiskpower198 going across 6 paths to the hdisk devices listed above.

Note that you won't be able to access the underlying child devices directly. Use the PowerPath devices if you wish to test the underlying devices. Alternatively, if you wish to test the child devices, use the powermt remove dev=xxxx to remove the association between the underlying device and PowerPath device.