Optimizing Java™ Technology Software Performance on HP OpenVMS

TIPS AND TRICKS FOR USERS

July 2003

This document contains usage information to optimize performance of the Software Development Kit (SDK) for the OpenVMS Alpha Operating System *for the Java™ Platform.*

Operating System	HP OpenVMS Alpha Version 7.2-1 or higher
Software Version	Software Development Kit (SDK) v 1 3.1-1 (or higher) for the OpenVMS Alpha Operating System *for the Java™ Platform*

Hewlett-Packard Company
Nashua, New Hampshire

CONTENTS

Purpose of This Document

The purpose of this document is to present practical tips and guidance for optimizing the performance of SDK and SDK-based applications on OpenVMS Alpha.

The information available here consolidates the experience of OpenVMS engineers, drawing upon their hands-on SDK experience.

With this document, you have, for the first time, a systematic way to optimize your OpenVMS SDK environment and applications.

Key

$	**Setup-related**
∞	**Developing SDK applications**
☢	**Running SDK applications**
☣	**Troubleshooting SDK applications**

1. THE SDK CHECKLIST

Start by verifying you meet or exceed the SDK hardware and software requirements, then look at the questions under *Applications* and *Issues* to see if any apply to your situation.

Hardware *What kind of system(s) do you have?*	**PERFORMANCE**		
	Best	**Avg.**	**Low**
? **System speed**	Above 500MHz	500MHz	Below 500MHz
? **System memory**	1GB or higher	512MB	Minimum 256MB

Software

What software versions do you have?

? **OpenVMS Alpha**

✓ The current version of OpenVMS is 7.3-1. You must be using Version 7.2-1 or later.

? **SDK for the OpenVMS Alpha Operating System *for the Java*™ *Platform***

✓ The current version of the SDK is the Software Development Kit (SDK) v 1.4.1 for the OpenVMS Alpha Operating System *for the Java*™ *Platform*.
You must be using Version 1.3.1-1 or later.

Available from the Java Technology Software Download Page
http://h18012.www1.hp.com/java/download/index.html

? **C Run-Time Library**

✓ Use the latest available versions of the ECOs for your C RTL.

(On the Software Patches (ECOs) Download page, search for `DEC-AXPVMS-VMS<Version>_ACRTL` with the highest version number.)

As of June 2002, the names of the latest available ACRTL images are:

For OpenVMS 7.2-1
`DEC-AXPVMS-VMS721_ACRTL-V0400--4`
For OpenVMS 7.2-1H1
`DEC-AXPVMS-VMS721H1_ACRTL-V0300--4`
For OpenVMS 7.2-2
`DEC-AXPVMS-VMS722_ACRTL-V0100--4`
For OpenVMS 7.3
`DEC-AXPVMS-VMS73_ACRTL-V0200--4`

Available from the Software Patches (ECOs) Download Page
http://ftp.support.compaq.com/patches/.new/openvms.shtml

? **TCP/IP Services**

✓ You must be using 5.0A or later and mandatory patches.
The current version for OpenVMS is 5.3 ECO 1.

Information available from the TCP/IP Services for OpenVMS Page
http://h71000.www7.hp.com/network/tcpip.html

? **Operating System ECOs**

✓ These are required to install and run the SDK.

Follow instructions on the SDK v1.3.1-3 or 1.4.1 Patch Installation Page
http://h18012.www1.hp.com/java/download/ovms/1.3.1/sdk1.3.1_patches.html
http://h18012.www1.hp.com/java/download/ovms/1.4.1/sdk1.4.1_patches.html

Developing SDK Applications

Which of these descriptions characterize the SDK applications you are developing?

? **Use fork/exec subprocesses**

✓ See Optimizing Parent-Child Processes: Increase the mailbox value

? **Process data from child processes**

✓ See Optimizing Parent-Child Processes: Emulate UNIX pipes

Running SDK Applications

Which of these descriptions characterize the SDK applications you are running?

? **Execute on different systems with varying amounts of memory**

✓ See Managing Memory: Set heap values dynamically

? **Run more slowly with the Fast VM than the "classic VM"**

✓ See Managing Memory: Memory usage

? **Rapidly recycle large objects**

✓ See Top Performance Tips: Shorten garbage collection

? **Use files exclusively on an ODS-5 volume**

✓ See Managing Memory: Reducing file name mapping

? **Small client-side applications**

✓ See Managing Memory: Optimizing for client-side applications

Troubleshooting Your Application

Which of these descriptions characterize the issues are you experiencing?

? **High Log Name Translation Rate**

✓ See Other Performance Tips: Reduce non-user mode

? **High "Exec Mode"**

✓ See Other Performance Tips: Reduce non-user mode

? **Excessive paging or page faults with a J2EE server**

✓See Other Performance Tips: Increase heap values

? **Continuous polling for changes in monitored sets of files**

✓ See Other Performance Tips: Increase the caching interval

? **Excessive file name mapping**

✓ See Other Performance Tips: Reduce file name mapping

? **Excessive `stat()` calls**

✓ See Other Performance Tips: Avoid secondary stat() calls

? **Excessive file sharing**

✓ See Managing Memory: Dynamically set heap values

? **Large directory reads**

✓ See Other Performance Tips: Limit the number of versions

2. MANAGING MEMORY

Managing available system memory is the most effective means for maximizing SDK application performance. This section tells you how to optimize memory allocation on your Alpha system. The starting point is to begin with enough physical memory because the Java architecture, in particular the Java Virtual Machine (JVM), puts a very high burden on system resources.

Set appropriate process quotas

The JVM was designed to perform optimally on UNIX systems, where each process is given large quotas by default. On OpenVMS, the default behavior is for each process to have smaller quotas so that many processes can coexist on a system. To get the best SDK performance on OpenVMS, set process quotas to match a typical UNIX system. In most cases, these also represent the minimum quota settings.

✓Recommendation

The following numbers closely match the default UNIX process quota settings:

UAF FILLM	4096
CHANNELCNT	4096
WSDEF	2048
WSQUOTA	4096
WSEXTENT AND WSMAX	16384
PGFLQUO	2097152
BYTLM	400000
BIOLM	150
DIOLM	150
TQELM	100

Calculating CHANNELCNT

CHANNELCNT is a SYSGEN parameter, not a UAF quota, and therefore requires a reboot for any change to take effect. Note that the SDK process will have the *lower* value of the UAF quota FILLM or the SYSGEN parameter CHANNELCNT.

Calculating PGFLQUO

A good number for `PGFLQUO` is `2 x` heap size. For example, `128MB` `(2*128*1024*1024)/512 = 524288`. (The recommended minimum `PGFLQUO` is `96MB` when using the Runtime Environment (RTE). When you increase the `PGFLQUO` parameter, you should also increase the system's page file size as needed to accommodate the new `PGFLQUO` parameter.)

See also this section: Set heap values dynamically

Memory usage with the Fast VM

The Fast VM is optimized for large-memory systems – meaning that a number of tradeoffs have been made that favor speed of execution over memory usage. As a result, the Fast VM uses more physical and virtual memory for the same application - often as much as 50 percent more than the classic SDK JIT (“classic VM”). This can lead to excessive paging and degraded performance if the system is not tuned correctly or if there is insufficient physical memory on the system.

See also Other Performance Tips: Use the Fast VM

✓Recommendation
If you notice that your application runs more slowly with the Fast VM than the classic VM, you should do the following:

- Experiment with explicit `-Xmx` and `-Xms` values rather than letting the Fast VM pick the defaults.

See also this section: Set heap values dynamically

- Increase process quotas based on the amount of available physical memory.

See also this section: Set appropriate process quotas

- Monitor your system for excessive page faults.
- Monitor your system's free pages list to see if you need to add physical memory.

Rule of Thumb
If you are short on physical memory, you are better off with a smaller maximum heap (`-mx<n>m`) value. This will increase garbage collections, avoiding excessive page faults due to a large heap size.

See the OpenVMS Performance Management manual
http://h71000.www7.hp.com/doc/72final/6491/6491PRO.HTML

Set heap values dynamically (or not)

Rather than use fixed values for the default settings for the memory allocation pool (SDK heap), you can let the Fast VM determine the defaults for the initial heap size (`-Xms`) and the maximum heap size (`-Xmx`) dynamically based on the environment in which it is executing.

The Fast VM uses the following formula:

max_memory = *lesser of physical memory and total memory available to the process**
default initial heap size = *10% of max_memory*
default maximum heap size = *60% of max_memory*

* Physical memory size versus `PGFLQUO` (`WSMAX` does not figure in this calculation).

Therefore, the Fast VM adjusts the heap size based on the amount of memory that is available. This generally produces better results than specifying fixed `-Xmx` and `-Xms` values, especially for an application that is executed on different systems with varying amounts of memory.

✓Recommendation
Under some circumstances, you may be able to obtain better results by specifying your own `-Xmx` and `-Xms` values rather than letting the Fast VM pick the defaults. To determine what values to use, you should use the `-verbose:gc` command line option to monitor your application's heap activity.

If you notice that a large number of garbage collections are occurring in a short time space, increase the heap size as much as possible without causing excessive page faults.

But...
Never set the maximum memory size to greater than the available physical memory. This will adversely affect performance in all cases.

Optimize for client-side applications

The Fast VM is optimized for large, long-running programs running on server systems. If you want to use the Fast VM on workstations for client-side applications, you can reduce its resource requirements. This client configuration significantly reduces the Fast VM memory footprint.

✓Recommendation
Start the Fast VM using the `-client` switch. This is equivalent to setting the following switches:

```
java -Xmx64m -Xglobal128m -Xgc:compacting
```

You can also override the individual settings that make up the `-client` switch. For example: `java -client -Xmx256m` initializes the Fast VM with a maximum heap size of `256MB`, with a maximum global region size of `128MB`, and with the compacting collector.

Eliminate paging with J2EE application servers

If you are running a J2EE application server, such as BEA WebLogic Server, you will realize better performance if you can eliminate paging. Paging is part of the normal system operation, but paging when running a J2EE application server can be detrimental.

✓Recommendation
Setting `WSMAX` or `WSEXTENT` too low produces page faults. Use the following formula to set `WSMAX(SYSGEN)` and `WSEXTENT(AUTHORIZE)` correctly:

```
MaxJavaHeapSize + MaxJavaHeapSize * 20% Divided By 8KB Pages
```

Example
If `MaxJavaHeapSize` is `–X-mx512m (512MB` of memory), then:

```
(512MB + 512MB * 0.2)/8192B = 78643 (8KB pages)
```

But…
It is essential that physical memory *be greater than* heap size. If the heap is not allowed enough physical memory, the application will page heavily. This paging will result in a substantial performance reduction.

Use `$ MONITOR SYSTEM` to monitor your system's page fault rate.

Reduce file name translation (mapping) overhead dynamic

The RTE is designed to map file names that are not directly representable on an OpenVMS file system into ones that are representable. This file name mapping requires several (potentially large) buffers on the stack. As the number of requested mappings increases, an increasingly large number of such buffers are required.

(The latest C RTL ECO enriches the kinds of file names you can directly specify on an ODS-5 volume under OpenVMS Alpha.)

✓Recommendation
If your application uses files exclusively on an ODS-5 volume and your system has the latest C RTL ECO, you can probably reduce the overhead of file name mapping. Whenever you eliminate a file name mapping option, you reduce the number of internal buffers that need to be allocated - leading to a smaller memory footprint for your application.

When no filename mapping is needed (defining `JAVA$FILENAME_CONTROLS` to be 0), you can lower the stack size (-`Xss<number>`) value. You should realize gains in memory usage and thread creation.

For more details about using JAVA$FILENAME_CONTROLS,
see Other Performance Tips: File Tuning: Reduce file name mapping

3. OTHER PERFORMANCE TIPS

This section presents other performance tips that are commonly effective when troubleshooting your application. You should consider each of these and determine if they are applicable to your situation.

GENERAL TUNING

Set up the Fast VM

✓Recommendation

For the best performance, you should use the Fast VM (virtual machine) unless you have compelling evidence that your application is required to use the "classic VM."

The Fast VM is optimized for SDK runtime performance on OpenVMS Alpha systems. (Starting with SDK v1.3.1-2, the Fast VM is included with the SDK kit.)

The Fast VM is *not the default* VM**.** You must set up the Fast VM.

To set up the SDK environment with the Fast VM
Use the `FAST` parameter with the following command:

```
$ @SYS$COMMON:[JAVA$131.COM]JAVA$131_SETUP FAST
```

Enter the `java -version` command to see a message like the following:

```
$ java -version
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition
Fast VM (build 1.3.1-n ...)
```

where *n* identifies the specific SDK point release of the version that is installed, and `Fast VM` identifies the virtual machine.

Note To revert to the classic VM, type the following command:

```
$ @SYS$COMMON:[JAVA$131.COM]JAVA$131_SETUP
```

See SDK Release Notes: Using the Fast VM
http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#usingfastvm
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#usingfastvm

Reduce non-user mode

If you are running high-resource applications on multiple-CPU systems, try reducing the amount of CPU time in *non-user* mode. Two situations apply:

Is your system experiencing a high Log Name Translation Rate?
(To observe this, run: `$ MONITOR IO`)

✓Recommendation

Try enabling the `DECC$ENABLE_GETENV_CACHE` logical. This logical will allow the C RTL to cache the entries returned by the `getenv()` function, reducing the number of calls to `sys$trnlmn()`, and allowing `getenv()` to work in user mode versus non-user mode.

But…
If you enable this logical, you will need to restart the application if you change any logical name.

Is your system experiencing a high "Exec Mode"?
Exec Mode is defined to mean a combination of Kernel, Executive, and Supervisor modes. To observe your system's statistics for these, type the following command:

```
$ MONITOR SYS/ALL
```

Some applications spend considerable time creating files, monitoring directories for the existence of files, and testing properties of files using file methods. This continuous polling for changes in a monitored set of files can result in hundreds of unnecessary calls per second to the `stat()` function.

Example
This shows the monitor statistics for a system where almost one third of CPU cycles are spent in exec mode (kernel, executive, and supervisor mode).

```
OpenVMS Monitor Utility
                              SYSTEM STATISTICS
                                   on node                    ~33%
                           18-APR-2002 21:13:12.32

                                    CUR         AVE         MIN         MAX

     Interrupt State               4.83        3.71        0.00        7.16
     MP Synchronization            0.16        0.08        0.00        0.33
     Kernel Mode                  26.50       20.90        0.00       37.00
     Executive Mode               13.50       11.48        0.00       24.83
     Supervisor Mode               0.66        0.36        0.00        0.83
     User Mode                    25.00       20.84        0.00       39.33
     Compatibility Mode            0.00        0.00        0.00        0.00
     Idle Time                   129.33      142.59       99.83      200.00
     Process Count                57.00       57.28       57.00       58.00
     Page Fault Rate               0.00        8.38        0.00      110.83
     Page Read I/O Rate            0.00        0.55        0.00       13.66
     Free List Size           232502.00   232746.15   232111.00   233402.00
     Modified List Size        47748.00    48753.55    47748.00    50740.00
     Direct I/O Rate               0.83        9.01        0.00      176.66
     Buffered I/O Rate           797.00      621.12        1.00     1072.16
```

✓Recommendation
If your application does these kinds of operations, you may benefit from the logical `JAVA$CACHING_INTERVAL`, which caches the information gathered by `stat()`.

See this section: Increase the caching interval

Build your classpath according to the number of classes.

Although it is typical to build a classpath based on a directory listing, if you can order your classpath from the highest number of classes found to the least number of classes found, this will improve performance.

✓Recommendation
Put the files with the largest number of classes first in your classpath.

Example
If you have a file `zz.jar` that contains 300 classes and a file `special.jar` that contains only 20 classes, put `zz.jar` first on the classpath. Specify the following:

```
$ define JAVA$CLASSPATH zz.jar, special.jar
```

When searching for class files, the JVM searches the classpaths as they are defined. So, in this case, if the JVM finds the class in `zz.jar`, the JVM will stop its search there.

Warning
Some applications require a specific order in the classpath. For example, if an application needs to find ECO classes before it finds the main classes, it will put the ECO jar file first and the main jar file second in the classpath.

Child processes

OpenVMS process creation is more resource-intensive than process creation on UNIX.

✓Recommendation
Do not create child processes if the function can be done with the SDK – meaning, within the same process space.

Some web application servers can choose between creating a child process to compile from JSP-to-servlet format or compiling the class without creating a child process. If such an option exists, choose *not* to create a child process.

If you must use a child process, consider changing the default buffering scheme from mailbox-based to socket-based by setting `JAVA$FORK_PIPE_STYLE = 2`.

See Optimizing Parent-Child Processes: User JAVA$FORK_PIPE_STYLE section

Reduce garbage collection times

Some applications that rapidly recycle large objects (such as complex nested structures and strings) can benefit from shorter garbage collection times. Other applications may also show performance gains using the non-default garbage collection scheme available with SDK v1.3.1-1 or later using the Fast VM.

Explanation
SDK v1.3.1-1 introduced an alternative garbage collection scheme for the Fast VM. The "compacting garbage collector" compacts live data in-place, rather than copying it, as the default collector does. Using a *mark-sweep/mark-compact* collection scheme, it avoids moving unnecessary data. You may realize better performance characteristics and lower heap-size requirements for applications in which the heap contains a high percentage of long-lived data. This collector can also perform minor collections, rather than collecting the entire heap, when the percentage of live data is moderate. It can also perform sweeps, which free space without moving any data at all. Therefore, it may also provide performance advantages for certain applications with a moderate amount of long-lived data, but a high rate of short-lived object turnover.

✓Recommendation
For those applications, you should try garbage compacting with the Fast VM by typing:

```
-Xgc:compacting
```

Note `-Xgc:copying` causes the Fast VM to use the default ("copying") collector.

FILE TUNING

Reduce file name mapping

The logical `JAVA$FILENAME_CONTROLS` determines the amount of file name mapping that the SDK does to fit UNIX-style file names on an ODS-2 file system. The default performance is to support ODS-2 and ODS-5 file structures.

✓Recommendation

For the best performance, use only the features (enabled bits in `JAVA$FILENAME_CONTROLS`) required by the application. To see individual filename mappings as they occur, define the logical `JAVA$SHOW_FILENAME_MAPPING`:

```
$ DEFINE JAVA$SHOW_FILENAME_MAPPING 1
```

To change the `JAVA$FILENAME_CONTROLS` default values, edit the following DCL file:

```
$ SYS$COMMON:[JAVA$131.COM]JAVA$FILENAME_CONTROLS.COM
```

Best performance

- Only use an ODS-5 disk and define `JAVA$FILENAME_CONTROLS = 0` or `1`

Also

- Use the latest C RTL ECO, which supports the following logicals:

```
DECC$ARGV_PARSE_STYLE
DECC$EFS_CASE_PRESERVE
DECC$EFS_CASE_SPECIAL
DECC$EFS_CHARSET
```

But…

The application should be expecting **100 percent** UNIX-style file names and file system.

See also Managing Memory: Reduce file name mapping

What file name mapping is right for your application?

Although the best performance is to disable any additional SDK translation, it might not be practical to disable all the file name mapping. You need to understand your application's requirements – for example, does it need to read files from an ODS-2 disk? If so, you will still need additional file name mapping. The following table shows available file mapping values.

File Name Mapping Table

Option	Value
Support UNIX and VMS filename	%x00000008
Support Dir in the filename	%x00000200
Support Valid characters in filename	%x00001000
Support Hidden Filename (replace with _)	%x00004000
Support Hidden Filename (remove the .)	%x00008000
Support Multi dot in directory (replace with _)	%x00020000
Support Multi dot in directory (remove dots)	%x00040000
Support Multi dot in file, keeping last	%x00100000
Support Multi dot in file, keeping first	%x00200000
Support more than 39 characters by truncation	%x04000000
Support more than 39 characters by moving the dot	%x08000000

Support Directory remapping	%x20000000

Review the description of each setting in the SDK Release Notes to understand what setting or combination of settings is likely to best suit your application.
http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#unix_style
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#unix_style

Increase the caching interval

Some applications spend considerable time creating files, monitoring directories for the existence of files, and testing properties of files using file methods such as:

```
File file = new File(name);
file.exists()
file.isDirectory()
file.isAbsolute()
```

These methods indirectly use the underlying C RTL `stat()` function.

A typical sequence might appear as:

```
if (file.exists()) { // ends up calling stat()
if (file.isDirectory()) // ends up calling stat() again
             ...
             }
             }
```

Some web servers have been observed making hundreds of calls per second to monitor the status of the same group of files. In such cases, increasing the caching interval of the overall speed-up from eliminating continuous polling in such cases can be dramatic.

✓Recommendation

If your application does these kinds of operations - continuous polling for changes in a monitored set of files – and you are using SDK v1.3.1-3, try this optimization.

By default, this optimization is not enabled. You can enable it by setting the `JAVA$CACHING_INTERVAL`[1] logical to a positive non-zero value that represents the caching interval.

For example, to set a 1 minute caching interval, you would use:

```
$ DEFINE/JOB JAVA$CACHING_INTERVAL 60
```

The information gathered by the `stat()` function is cached for the interval indicated before it is deemed stale and the cache is invalidated.

Hence, in the code fragment above, the inquiry `file.isDirectory()` is answered from cached information and no additional I/O is actually performed. This represents a considerable difference in time.

Usage Notes

The cache is invalidated and a real call to `stat()` is made the first time after:

- The current caching interval expires.
- An explicit action is taken within the current application that may be assumed to change the validity of the cached information, including:
  - A file is opened for something other than `READ`.
  - A file or directory is created or destroyed.

[1] `JAVA$CACHING_INTERVAL` is new to SDK 1.3.1-3

- A child process is spawned via `Runtime.exec()` or completes.

The main drawback to reduced caching is that, if another application, without the cooperation of your application (for example, FTP), copies a file into the monitored set, the RTE might not see it until the cache is validated for one of the reasons above.

Avoid secondary `stat()` calls

Because of the way file name mapping works, and the way in which directories that contain dots are handled by OpenVMS Alpha, the RTE may need to do an extra `stat()` function call whenever you seek a class file that is not found on the initial `stat()` call.

Example
Suppose you import an application from another platform that contains a directory named `project-3.1-A` within its overall directory structure. During operation you may need to verify that the directory exists. If you are using ODS-2, you cannot represent a directory named `project-3.1-A`. As a result, if the application checks that the directory exists, the RTE first tries to access a directory that does not exist: `[projects.project-3.1-a]`. When that fails, it does another `stat()` trying a modified path by removing all dots from directory names: `[projects/project-3_1-A]`.

Explanation
The second attempt is needed to support the OpenVMS scheme for remapping of directories that contain dots. In general, for whatever reason a `stat()` fails, the RTE attempts another `stat()` call with a potentially modified path name. You can use a logical to prevent this unwanted behavior.

What is the impact of doing a second `stat()` call?

Modern SDK applications are typically constructed by drawing together class files (libraries) provided by a number of vendors. In order to tie this together at runtime, you must specify long classpath strings that can contain hundreds of individual path names.

For example, for projects/classes/MyApp.class, the RTE would try to stat() `[projects.classes]MyApp.class` first. If for any reason that fails, it would try `[projects.classes]MyApp_class`.

When an application is looking for a particular class file, it will generally need to look through approximately 50 percent of the paths to find it; this means that the RTE will do two `stat()`'s for every class file lookup failure.

Each time the RTE looks up a path that does not contain the class it is seeking, the failure of the first `stat()` causes a secondary `stat()` to be issued. If you multiply these secondary `stat()` calls by the several hundred classes that are loaded during a typical application startup, you can see that the total number of secondary `stat()` calls can account for a significant amount of application startup time.

✓**Recommendation**

If you know your application will access ODS-5 format exclusively—or you know that your application *never* deals with directories that have dots in their names—you can disable the secondary `stat()` call.

Use the `JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT` logical to avoid secondary `stat()` calls:

```
$ DEFINE/NOLOG JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT TRUE
```

Restrict file sharing

Most SDK applications expect UNIX file-sharing mode, but for the best performance you should enable only the file-sharing level needed for the application:

✓**Recommendation**
Set file sharing according to the following scale (from best performance to best feature):

1. **Best Performance – average sharing**
   `DECC$FILE_SHARING ENABLED`
   Files are open for sharing (best for performance and sharing, recommended)
2. **Above-Average Performance – above-average sharing**
   `JAVA$FILE_OPEN_MODE 3`
   Files are open for sharing with additional RMS buffer flushing
3. **Average or Below-Average Performance – best sharing mode**
   `JAVA$FILE_OPEN_MODE 2`
   Every read/write is synchronized with the disk (slowest performance)

(**No sharing**
`JAVA$FILE_OPEN_MODE 0`
No file sharing is *not* recommended for most applications.)

See also Using Logicals: `JAVA$FILE_OPEN_MODE`

Disable case logical

Because the SDK is case-sensitive and needs to support ODS-2 and ODS-5 disk formats, the `File.list()` function will open and read any file names that end with `.java` and `.class`, trying to match up the "real" SDK or class name. This will slow down an application with large `.java` or `.class` directories.

✓**Recommendation**
If you are using an ODS-5 disk, set `JAVA$READDIR_CASE_DISABLE` to `True`. Use of this undocumented logical prevents a behavior that is not required on an ODS-5 disk and can be costly to performance.

Also…
You must also define `JAVA$FILENAME_CONTROLS = 0` or `1`

See *this section: Reduce File Name Mapping: Best performance option.*

Limit the number of versions

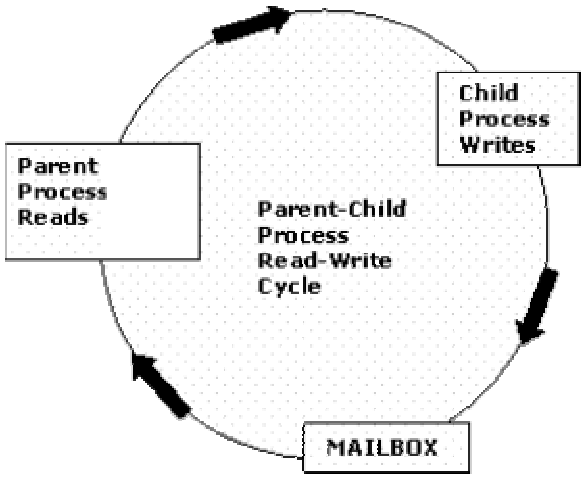
Many applications use the SDK Swing File Chooser, which reads an entire directory and calls the `File.isDirectory` method for each item in the directory. Some applications have their own functions that verify the existence of files. For example, NetBeans' `file_exists` method (which takes a string and then reads the directory list until it finds a string match). All such methods are slowed down by multiple file versions.

✓**Recommendation**
The more you limit the number of files in a directory (by eliminating old file versions), the better your SDK will performance will be.

4. OPTIMIZING PARENT-CHILD PROCESSES

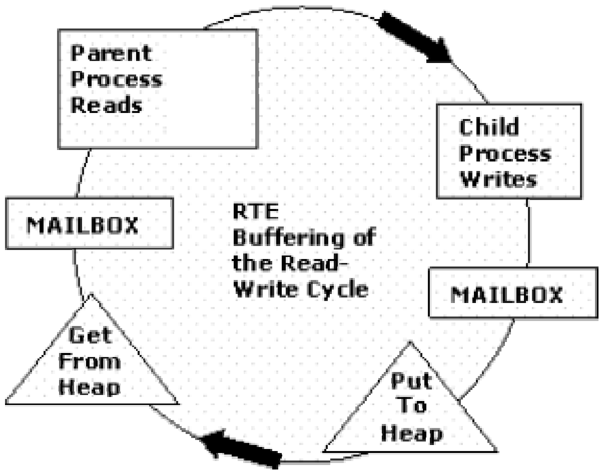
When developing SDK applications, begin with a clear understanding of whether or not all processes can execute in the SDK memory space or whether you will need non-SDK processes.

On OpenVMS Alpha systems, SDK process creation and parent-child relationships are managed by native threads. If they pass data between them it is done by *mailboxes*. For example, a typical scenario is where a child process (the mailbox *producer*) produces a stream of output, and the parent process (the mailbox *consumer*) consumes this stream as input.



Some applications are written to be synchronous, so that the parent process starts the child process and then waits for the child process to finish before starting to read its stream of data. Other applications are written to be asynchronous, so that the parent starts reading as soon as it can but may not keep up with the child process. Both of these approaches can get into trouble because the mailboxes are of a finite size.

If the mailbox fills up, the producer is blocked from doing further work. Meanwhile, the consumer is waiting for the producer to finish and also cannot proceed. This state is known as a Resource Wait: MailBoX (`RWMBX`) wait state (Resource Wait: INner mode Semaphore, also known as `RWINS`), and in this state the process running the application is blocked.

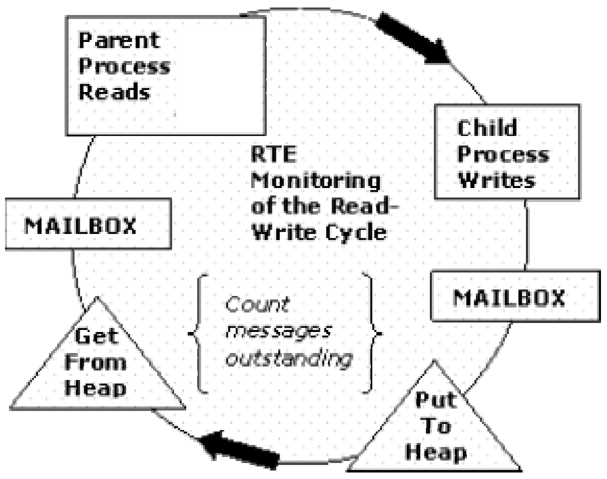


The RTE attempts to avoid this situation from arising by buffering between the mailbox and the mailbox consumer. One thread reads data from the mailbox and stores it in heap storage. Another thread reads buffered data from this heap and delivers it to the parent process when it asks for it.

In the case where the parent process is waiting for the child process to finish, this buffering may empty the mailbox enough so that the producer can continue to run and finish, thereby freeing the consumer to continue as well. In the case where the parent process is continually reading but is temporarily being overwhelmed, the buffering smoothes the production-consumption data flow.

However, if the parent process is waiting for the child process to complete before consuming any of the information stream, this buffering mechanism itself starts to stall when the volume of output exceeds the amount of heap storage available. When no more heap is available, `PutToHeap` cannot empty the mailbox, again resulting in an `RWMBX` wait state. The blocked state persists unless some asynchronous agent comes along and changes the resources involved, such as releasing some heap storage.

In most cases, however, the added capacity of heap storage is enough for the child to complete and the parent to continue. As a further enhancement to control this data flow, the RTE monitors the pipeline for how many messages are outstanding:



When the number of messages written exceeds the number read by an amount equal to the lesser of 1024 and the value of the logical `JAVA$FORK_MAILBOX_MESSAGES`, the RTE stops the write to prevent the `RWMBX` condition. `JAVA$FORK_MAILBOX_MESSAGES` is assumed to be 8 if not defined.

This temporary curtailment of production allows the parent thread to catch up (get within the exceeded message threshold), and the overall application continues.

Increase the Mailbox Value

You can tune SDK applications for process creation using the following logicals: `JAVA$FORK_MAILBOX_MESSAGES` and `JAVA$FORK_PIPE_STYLE` or `JAVA$EXEC_USE_PIPES`[2].

With the default `JAVA$FORK_MAILBOX_MESSAGES` value of 8, any time the producer gets 8 records ahead of the consumer, the parent buffer stream will be temporarily suspended.

[2] `JAVA$FORK_PIPE_STYLE` is the preferred logical for users of SDK 1.3.1-2 or later.

If your application produces many small records, you might want to set a value of `JAVA$FORK_MAILBOX_MESSAGES` to a larger value - something on the order of 1024 divided by your typical mailbox record size.

The bigger you can make this value - think of it as a gating factor - the more efficient the operation becomes because you are minimizing the start-stop time of the producer thread. However, if you make it too large, you might induce the `RWMBX` state. If you see the SDK application in `RWMBX` state for a period of time without having explicitly used `JAVA$FORK_MAILBOX_MESSAGES`, try setting it to a value less than the default value of 8.

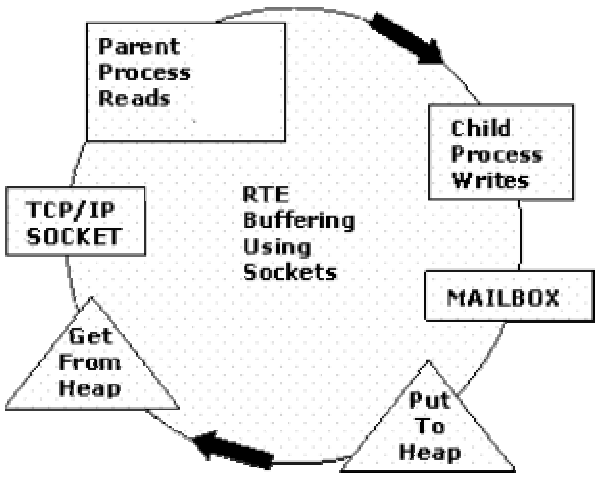
Using JAVA$FORK_PIPE_STYLE

With Version 1.3.1-2 of the SDK, a new logical, `JAVA$FORK_PIPE_STYLE`, was introduced. This logical extends the behavior of `JAVA$EXEC_USE_PIPES` with an optional non-mailbox buffering scheme. Note that the values used for `JAVA$FORK_PIPE_STYLE` are not equivalent to those for `JAVA$EXEC_USE_PIPES`:

JAVA$FORK_PIPE_STYLE

Value	Performance	Behavior
1	Average	Default mailbox behavior
2	Best	Uses a socket rather than a mailbox for buffering data between the heap and the SDK application parent process reads (requires a TCP/IP stack)

Defining `JAVA$FORK_PIPE_STYLE` with the value of 2 results in the use of a socket rather than a mailbox for buffering data between the heap and the SDK application parent process reads. This requires a TCP/IP stack.



This is the closest to simulating UNIX pipe functionality. Because the SDK application is reading from a socket device, it can use unbuffered I/O, `ioctl()` features, cancel I/O operations, and so on. A significant performance improvement should result when processing data from child processes.

5. USING SDK LOGICALS

This section presents usage information about SDK logicals that can help optimize your application but are not directly related to performance per se.

(Information about logicals that are performance-specific - such as `JAVA$CACHING_INTERVAL`, `JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT`, and `JAVA$FORK_MAILBOX_MESSAGES` – are in the previous sections.)

JAVA$DIRECTORY_MAPPING_NN

There is support for representing a directory structure that is more than 8 levels deep on an ODS-2 format disk. Not required with ODS-5.

If your application needs to use more than the OpenVMS maximum of 8 nested directories, define `JAVA$DIRECTORY_MAPPING_NN` to the required value.

Example

```
$ def java$directory_mapping_01 -
_$ "/foobar/redirect/testing/testing2/testing3=/foobar/cont123"
```

This tells the SDK that if it is opening a file called `data.txt`, for example, in `/foobar/redirect/testing/test2/testing3/` to open `/foobar/cont123/data.txt` instead.

JAVA$ENABLE_ROOT_WITH_000000

This is an incompatible change to behavior in SDK v1.3.1-1.

SDK v1.3.1-1 fixed a problem that occurred when you tried to open a file using rooted logicals such as:

```
   File f = new File ("/sys$sysroot");
```

If you attempted to get a directory listing of rooted logicals, (for example, `sys$sysroot`) or of disk names (for example, `/sys$sysdevice` or `/node$dka200`). With the current C RTL, the RTE must internally add a `/000000` to these names to make this directory listing operation work correctly.

This workaround created problems with nonrooted logicals like `/sys$errorlog`. HP expects that newer C RTLs will handle these special cases internally, and the RTE will no longer need to use this workaround. Anticipating this transition, the automatic addition of `/000000` is no longer the default behavior of the current C RTL. If you still need this capability in your application with the current C RTL you must explicitly request it by defining a logical:

```
   $ define/job JAVA$ENABLE_ROOT_WITH_000000 TRUE
```

JAVA$EXEC_TRACE

This is a new logical to help debug `Runtime.exec()` function calls.

The logical name `JAVA$EXEC_TRACE` is available to help debug `Runtime.exec()` calls on OpenVMS. When this logical is defined,

```
      $ define/job JAVA$EXEC_TRACE true
```

a printout displays the C RTL exec variant and its list of arguments.

JAVA$FILE_OPEN_MODE

The RTE supports limited file-sharing and has several user-selectable modes of file sharing. To enable RTE file-sharing modes, define the logical `JAVA$FILE_OPEN_MODE` to any of the following values:

`0` (or not defined)	No file sharing
`2`	Synchronous writes – for every write to a file, the file is in sync with the disk. **Note** This mode gives you file sharing close to what the SDK expects. However, processes that normally take seconds can require minutes.
`3`	A table of all open file descriptors is kept by the SDK. The SDK then monitors when an application writes to a file and sets a `write_pending` flag. Before every read operation the SDK scans the list of open file descriptors. If a match is found with a write pending, the pending write is flushed to the disk before the read.

Note
`The DECC$FILE_SHARING` logical has replaced the value of 1.

JAVA$KEYBOARD_TYPE_DEC

By default, the RTE responds to keystrokes, assuming they came from a PC keyboard with the `NumLock`, /, * and - (`NUM_LOCK, DIVIDE, MULTIPLY`, and `SUBTRACT`) on the top row of the numeric keypad.

When you are using a DEC keyboard, which has different keys at the top of the keypad (`PF1`, `PF2`, `PF3`, and `PF4`), this interpretation will not be correct for those keys.

If you define the `JAVA$KEYBOARD_TYPE_DEC` logical name as shown:

```
$ define/log JAVA$KEYBOARD_TYPE_DEC true
```

the keys will be interpreted correctly for the DEC keyboard. In addition to the keys mentioned, the `Find` and `Select` keys will behave as expected for a DIGITAL-style keyboard.

JAVA$OMIT_CASE_CHECK

The case of `JAVAC` and `JAR` file operands is determined automatically. Previously, HP had added the capability for JAVAC and JAR to accept wildcarded file specifications and would automatically determine the right case of the name. This allowed you to say:

```
$ JAVAC *.java
and
$ JAR cvf test.jar *.class
```

The SDK 1.3.1 extends this capability in these two tools for any `.java` and `.class` file operand, so you can say, for example:

```
$ JAR cvf test.jar TESTPLOT.CLASS
```

and you need not be concerned about the case of mixed-case nature of the name:

```
$ JAR cvf test.jar "TestPlot.class"
$ JAR cvf test.jar "TESTPLOT.CLASS"
or
$ JAR cvf test.jar "testplot.class"
```

This can be particularly helpful to those using DCL to automatically generate operands, because DCL will uppercase them. In general, this reduces the number of situations where you need to present the exact-case name and quote it.

Note
This works only for `JAVAC` and `JAR`. You still need to use the right case for verbs like `JAVA` itself.

This feature is enabled by default; if it causes problems with existing command procedures, it can be entirely disabled by setting:

```
   $ define JAVA$OMIT_CASE_CHECK true
```

JAVA$SHOW_FILENAME_MAPPING

To see individual filename mappings as they occur, define the logical `JAVA$SHOW_FILENAME_MAPPING`:

```
      $ DEFINE JAVA$SHOW_FILENAME_MAPPING 1
```

This is useful if you are experiencing problems with the way file names are being mapped, perhaps resulting in unexpected "File not found" messages.

VAXC$PATH

Before SDK v1.3.1-2, if you did not specify a classpath directory path, RTE would search only in the default directory. Beginning with SDK v1.3.1-2, you can use `VAXC$PATH` as an OpenVMS search path for locating `.EXE` or `.COM` files outside the default directory. For example,

```
$ define VAXC$PATH GNU:[bin],[],sys$common:[java$131.bin]
                        ! open source GNU files
   br = new BufferedReader(new InputStreamReader(

   rt.exec("chmod").getInputStream()));
```

The above will now search the three directories defined in `VAXC$PATH` for `chmod.`, `chmod.exe`, or `chmod.com`. Also:

```
   br = new BufferedReader(new InputStreamReader(

   rt.exec("javac").getInputStream()));

   The above will search for javac., javac.exe, or javac.com.
```

JAVA$CACHING_ATEXIT_PRINT_STAT

This undocumented logical dumps the `stat()` caching indexes after the program exits. To be used for troubleshooting `JAVA$CACHING_INTERVAL:`

```
      Value 100   Dump all information
      Value 1     Dump summary
```

JAVA$CREATE_DIR_WITH_OWNER_DELETE

This undocumented logical is replaced by `DECC$file_owner_unix`

JAVA$CREATE_ONE_VERSION

This undocumented logical creates only one version of a file.

JAVA$RENAME_ALL_VERSIONS

This undocumented logical renames all versions of a file. The C RTL usually only renames the highest version of a file.

JAVA$WAIT_FOR_CHILDREN XX

This undocumented logical allows the SDK to wait `XX` seconds before calling `exit()`. There can be times when a child process is in the middle of a rundown (closing files) when the parent calls an `exit()`. This could result in an `ACCVIO` or `pthread` dumps.

6. OPTIMIZING APPLICATIONS

SDK applications come in many shapes and sizes. Each one has specific characteristics and requirements. This section presents some performance recommendations for four OpenVMS layered products.

BridgeWorks

These recommendations apply when generating SDK components on the same Alpha system as the wrapped application:

Initial heap size

Full memory allocation for complex structures may require a boost in the initial SDK heap memory size. For example, with SDK v1.3.1, the command `java -Xms64M -Xmx256 helloworld` allocates 64MB for minimum heap size and 256MB as the maximum. (Consult the `java -X` options on how to set these options.)

> **But…**
> Since a BridgeWorks solution is part-SDK and part native code, finding the right balance is essential - too much memory for SDK could degrade overall performance.

Increase heap values

Use very large `MX` and `MS` values to avoid letting the SDK slow-start its heap.

> **But…**
> If you allocate an initial heap size that is greater than your physical memory size, it may force the operating system to allocate virtual memory (temporarily paging data to the disk - a huge performance loss).

CSWS_JAVA

Finding the right quota balance depends on satisfying both the Secure Web Server and the CSWS_JAVA module.

Tomcat and JServ requirements

When you select the user account for the Jakarta (Tomcat) or JServ servlet engines, consider SDK quota requirements to ensure best performance of your SDK applications.
The default quota values for the `APACHE$WWW` account that are set by the HP Secure Web Server installation might not be optimized for the SDK. In particular, you might need to increase `FILLM` (and the related `CHANNELCNT SYSGEN` parameter), `PGFLQUO`, and `BYTLM`.

These are pooled quotas. If you are configuring the JServ servlet engine, which is a subprocess, you need to be aware of the impact on these quotas from other Apache child processes in the same job tree. The Jakarta (Tomcat) servlet engine is a detached process and is not affected by Apache child processes.

Faster updates to the JServ log file

The `JSERV.LOG` file remains empty until sufficient data causes the buffer to be flushed. (If the server is shut down, the log file is empty.) If, for example, you are debugging a servlet and you need more timely updates to the JServ log file, you can change the mode for flushing and sharing files. To do this, modify the logical `JAVA$FILE_OPEN_MODE` in the file `START_JSERV_MANUAL.COM` (located in `APACHE$COMMON:[000000]`).

You can define `JAVA$FILE_OPEN_MODE` to one of the following:

`0`	No file sharing – the default
`2`	Synchronous writes – for every write to a file, the file is in sync with the disk.
`3`	The JVM monitors when an application writes to a file and flushes writes to the disk before each read operation; however, only one process can share the file reliably.

But…
Do not define `JAVA$FILE_OPEN_MODE` to a value of `2` in a production environment. It will have an adverse effect on performance.

See also Other Performance Tips: File Tuning: Restrict File Sharing

NetBeans

There are defaults chosen in the NetBeans Launcher for the initial stack size, initial heap size, and maximum heap size. These values were selected based on tests that simulated an average workload.

If your NetBeans process is running out of heap or stack space, you may need to start NetBeans with a larger value for the stack size or maximum heap size.

This is an example of setting the maximum heap size to 350MB and the stack size to 2MB.

```
NetBeans "" "-Xmx350m –Xss2m"
```

BEA WebLogic Server

The `weblogic600` or `weblogic610` account should have the following minimum quotas:

PGFLQUO	1500000
WSDEF	8000
WSEXTENT	32000
WSQUO	16000
BYTLM	300000
FILLM	1024

Appendix A: Java$ Logicals

See SDK *Release Notes* (v 1.4.0 or earlier) or *User Guide* (v 1.4.1 or later): Using the Fast VM
For v 1.4.1:
http://h18012.www1.hp.com/java/documentation/1.4.1/ovms/docs/user_guide.html#usingfastvm
For v 1.3.1-6:
http://h18012.www1.hp.com/java/documentation/1.3.1-6/ovms/docs/release_notes.html#usingfastvm

Logicals that are referenced *only* in this guide are considered "undocumented." They are not supported - that is, you may use them at your own risk.

Logical Name	New in SDK 1.3.1-6 or 1.4.1	Documented	Web-based Release Notes References
JAVA$CACHING_ATEXIT_PRINT_STAT			Not supported
JAVA$CACHING_INTERVAL		✓	1.3.1.6, 1.4.1
JAVA$CLASSPATH		✓	1.3.1.6, 1.4.1
JAVA$CREATE_DIR_WITH_OWNER_DELETE	✓	✓	1.3.1.6, 1.4.1
JAVA$CREATE_ONE_VERSION			Not supported
JAVA$DAEMONIZE_MAIN_THREAD	✓	✓	1.3.1.6, 1.4.1
JAVA$DELETE_ALL_VERSIONS	✓	✓	1.3.1.6, 1.4.1
JAVA$DIRECTORY_MAPPING_COUNT		✓	1.3.1.6, 1.4.1
JAVA$DIRECTORY_MAPPING_NN		✓	1.3.1.6, 1.4.1
JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT		✓	1.3.1.6, 1.4.1
JAVA$ENABLE_ENVIRONMENT_EXPANSION	✓	✓	1.3.1.6
JAVA$ENABLE_ROOT_WITH_000000		✓	1.3.1.6, 1.4.1
JAVA$ENABLE_SIGQUIT_CTRLC	✓	✓	1.3.1.6, 1.4.1
JAVA$ENABLE_SIGQUIT_MAILBOX	✓	✓	1.3.1.6, 1.4.1
JAVA$EXEC_TRACE		✓	1.3.1.6, 1.4.1
JAVA$EXEC_USE_PIPES		✓	1.3.1.6, 1.4.1
JAVA$FILE_OPEN_MODE		✓	1.3.1.6, 1.4.1
JAVA$FORK_MAILBOX_MESSAGES		✓	1.3.1.6, 1.4.1
JAVA$FORK_PIPE_STYLE		✓	1.3.1.6, 1.4.1
JAVA$FSYNC_INTERVAL	✓	✓	1.3.1.6
JAVA$KEYBOARD_TYPE_DEC		✓	1.3.1.6
JAVA$OMIT_CASE_CHECK		✓	1.3.1.6, 1.4.1
JAVA$PRINT_COMMAND_ARGS	✓	✓	1.3.1.6, 1.4.1

Logical Name	**New in SDK 1.3.1-6 or 1.4.1**	**Documented**	**Web-based Release Notes References**
`JAVA$READDIR_CASE_DISABLE`	✓	✓	1.3.1.6, 1.4.1
`JAVA$RENAME_ALL_VERSIONS`	✓	✓	1.3.1.6, 1.4.1
`JAVA$SHOW_FILENAME_MAPPING`	✓	✓	1.3.1.6, 1.4.1
`JAVA$WAIT_FOR_CHILDREN`			Not supported
`VAXC$PATH`		✓	1.3.1.6, 1.4.1

Appendix B: DECC$ Logicals

The logical names in this listing are documented in the web-based *C Run-Time Library Reference Manual* and/or the current ACRTL ECO readme files.

ECO Key

4	dec-axpvms-vms721_acrtl-v0400--4
3	dec-axpvms-vms721h1_acrtl-v0300--4
2	dec-axpvms-vms73_acrtl-v0200--4
1	dec-axpvms-vms722_acrtl-v0100--4

Logical Name	New in ECO	Documented	Web-based Reference and ECO Information
DECC$ARGV_PARSE_STYLE	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$DEFAULT_LRL		✓	C RTL Reference
DECC$DEFAULT_UDF_RECORD		✓	C RTL Reference
DECC$DETACHED_CHILD_PROCESS		✓	C RTL Reference
DECC$DISABLE_POSIX_ROOT		✓	C RTL Reference
DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION	✓	✓	C RTL Reference 4, 3
DECC$EFS_CASE_PRESERVE	✓	✓	C RTL Reference 4, 3
DECC$EFS_CASE_SPECIAL	✓	✓	C RTL Reference 4, 3
DECC$EFS_CHARSET	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$EFS_FILE_TIMESTAMPS		✓	C RTL Reference
DECC$ENABLE_GETENV_CACHE		✓	C RTL Reference
DECC$EXEC_FILEATTR_INHERITANCE		✓	C RTL Reference
DECC$FILENAME_UNIX_NO_VERSION	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$FILENAME_UNIX_ONLY	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$FILENAME_UNIX_REPORT	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$FILE_OWNER_UNIX		✓	C RTL Reference
DECC$FILE_PERMISSION_UNIX		✓	C RTL Reference
DECC$FILE_SHARING	✓	✓	C RTL Reference 4, 3
DECC$FIXED_LENGTH_SEEK_TO_EOF		✓	C RTL Reference

Logical Name	New in ECO	Documented	Web-based Reference and ECO Information
DECC$LOCALE_CACHE_SIZE		✓	C RTL Reference
DECC$MAILBOX_CTX_STM		✓	C RTL Reference
DECC$PIPE_BUFFER_SIZE		✓	C RTL Reference
DECC$POSIX_SEEK_STREAM_FILE	✓	✓	C RTL Reference 4, 3
DECC$READDIR_DROPDOTNOTYPE		✓	C RTL Reference
DECC$READDIR_KEEPDOTDIR			Not supported
DECC$RENAME_NO_INHERIT	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$SELECT_IGNORES_INVALID_FD	✓	✓	C RTL Reference 4, 3
DECC$SET_CHILD_STANDARD_STREAMS		✓	C RTL Reference 1
DECC$STDIO_CTX_EOL		✓	C RTL Reference
DECC$STRTOL_ERANGE		✓	C RTL Reference
DECC$TRACE			Not supported
DECC$THREAD_DATA_AST_SAFE	✓	✓	C RTL Reference 4, 3, 2, 1
DECC$TZ_CACHE_SIZE		✓	C RTL Reference
DECC$UMASK		✓	C RTL Reference
DECC$UNIX_PATH_BEFORE_LOGNAME		✓	C RTL Reference
DECC$USE_RAB64		✓	C RTL Reference
DECC$V62_RECORD_GENERATION		✓	C RTL Reference
DECC$VALIDATE_SIGNAL_IN_KILL		✓	C RTL Reference
DECC$XPG4_STRPTIME	✓	✓	C RTL Reference 4, 3

DECC$ Logicals by Function

The following table (from the C Run-Time Library Reference Manual) organizes most documented DECC$ logicals according to function and shows the default setting for each one.

Performance Optimizations	
DECC$ENABLE_GETENV_CACHE	DISABLE
DECC$LOCALE_CACHE_SIZE	0
DECC$TZ_CACHE_SIZE	2
Legacy Behaviors	
DECC$V62_RECORD_GENERATION	DISABLE
DECC$XPG4_STRPTIME	DISABLE
DECC$THREAD_DATA_AST_SAFE	DISABLE
File Attributes	
DECC$DEFAULT_LRL	32767
DECC$DEFAULT_UDF_RECORD	DISABLE
DECC$FIXED_LENGTH_SEEK_TO_EOF	DISABLE
Mailboxes	
DECC$MAILBOX_CTX_STM	DISABLE
Changes for UNIX Conformance	
DECC$STRTOL_ERANGE	DISABLE
DECC$VALIDATE_SIGNAL_IN_KILL	DISABLE
DECC$SELECT_IGNORES_INVALID_FD	DISABLE
General UNIX Enhancements	
DECC$ARGV_PARSE_STYLE	DISABLE
DECC$PIPE_BUFFER_SIZE	512
DECC$STDIO_CTX_EOL	DISABLE
DECC$USE_RAB64	DISABLE
Enhancements for UNIX-Style File Names	
DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION	DISABLE
DECC$EFS_CHARSET	DISABLE
DECC$FILENAME_UNIX_NO_VERSION	DISABLE
DECC$FILENAME_UNIX_REPORT	DISABLE
DECC$READDIR_DROPDOTNOTYPE	DISABLE
DECC$RENAME_NO_INHERIT	DISABLE
Enhancements for UNIX-Style File Attributes	
DECC$EFS_FILE_TIMESTAMPS	DISABLE
DECC$EXEC_FILEATTR_INHERITANCE	DISABLE
DECC$FILE_OWNER_UNIX	DISABLE
DECC$FILE_PERMISSION_UNIX	DISABLE
DECC$FILE_SHARING	DISABLE
UNIX Compliance Mode	
DECC$FILENAME_UNIX_ONLY	DISABLE
DECC$DETACHED_CHILD_PROCESS	DISABLE
New Behaviors for POSIX Conformance	
DECC$POSIX_SEEK_STREAM_FILE	DISABLE
DECC$UMASK	RMS default

File Name Handling	
DECC$READDIR_KEEPDOTDIR	DISABLE
DECC$EFS_CASE_PRESERVE	DISABLE
DECC$EFS_CASE_SPECIAL	DISABLE
DECC$UNIX_PATH_BEFORE_LOGNAME	DISABLE
DECC$DISABLE_POSIX_ROOT	DISABLE