

# VSI OpenVMS

## C Run-Time Library (C RTL) Release Notes

Publication Date: July 2023

**Operating System and Version:** ECO patch kit RTL V8.0

---

## C Run-Time Library (C RTL) Release Notes



VMS Software

---

Copyright © 2023 VMS Software, Inc. (VSI), Boston, Massachusetts, USA

### Legal Notice

Confidential computer software. Valid license from VSI required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for VSI products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. VSI shall not be liable for technical or editorial errors or omissions contained herein.

HPE, HPE Integrity, HPE Alpha, and HPE Proliant are trademarks or registered trademarks of Hewlett Packard Enterprise.

Intel, Itanium, and IA-64 are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group.

<b>C RTL Changes in ECO Patch Kit RTL V8.0 .....</b>	<b>1</b>
1. New Functions .....	2
2. Updates to Functions .....	6
3. Bug Fixes .....	7
4. New Header Files .....	8
5. Known Limitation .....	8
6. Documentation Update .....	8



# C RTL Changes in ECO Patch Kit RTL V8.0

This kit provides additional C RTL functions, updates to some functions, bug fixes, new header files, and identifies a known limitation. This ECO level is cumulative and includes the changes from all previous versions of C RTL ECO. Please note that C RTL ECO V7.0 was previously released as part of VSI OpenVMS x86\_64 V9.2-1. All changes through C RTL ECO V8.0 are included in this kit.

The ECO patch kit RTL V8.0 may be applied to the following VSI OpenVMS versions:

- VSI OpenVMS Integrity Versions 8.4-2L1 and 8.4-2L3
- VSI OpenVMS Alpha Versions 8.4-2L1 and 8.4-2L2

---

## Note

If you develop an application on a system with the RTL C99 or any later kit installed and intend it to be run on a system without those kits, you must compile your application with the switch `/DEFINE=(__CRTL_VER_OVERRIDE=80400000)`.

---

## Possible errors when compiling applications

It is possible that applications may incur compilation errors if the applications include definitions that conflict with the definitions now provided in the system header files. For example, if an application contains a definition of `int64_t` that differs from the definition included in `STDINT.H`, the compiler generates a `%CC-E-NOLINKAGE` error.

One solution is to remove the application-specific definition if the system-provided definition provides the proper functionality. To diagnose such problems, compile the application using `/LIST/SHOW=INCLUDE` and then examine the listing file.

There are different ways to resolve such problems:

- Remove the application-specific definition if the system-provided definition provides the proper functionality.
- Undefine the system-provided definition before making the application-specific definition. For example:

```
#ifdef alloca
#undef alloca
#endif
<application-specific definition of alloca>
```

- Guard the application-specific definition. For example:

```
#ifndef alloca
<application-specific definition of alloca>
#endif
```

## Manipulating Variable Argument Lists on x86-64

The implementation of variable argument lists on x86-64 is different than on Integrity and Alpha and may require source code changes, depending on how the lists are used.

On Integrity and Alpha, it is possible to copy one variable argument list to another using an assignment operator. For example:

```
va2 = va1
```

On x86-64, this does not work. Use the `va_copy` function for this purpose. For example:

```
va_copy (va2, va1)
```

On Integrity and Alpha, it is also possible to reference specific entries in the variable argument list using the subscript notation. For example:

```
int arg2 = va[1]
```

On x86-64, this does not work. Use the `va_arg` function for this purpose. For example:

```
int arg2 = va_arg(va, int)
```

## Online Help

The OpenVMS CRTL Help Library has been updated with the changes from several previously released ECO RTL patch kits, including the ECO patch kit RTL V8.0.

# 1. New Functions

This section describes the new C RTL functions introduced in the ECO patch kit RTL V8.0.

## alloca

### Format

```
#include <alloca.h>
void *alloca (unsigned int size);
```

### Description

The `alloca` function allocates `size` bytes from the stack frame of the caller. The memory is automatically freed when the function that calls `alloca` returns to its caller. See *VSI C User's Guide for OpenVMS Systems* for the `__ALLOCA` macro.

### Returns

The `alloca` function returns a pointer to the allocated memory.

## mempcpy

### Format

```
#include <string.h>
void *mempcpy (void *dest, const void *source, size_t size);
```

### Function Variants

The `mempcpy` function has variants named `_mempcpy32` and `_mempcpy64` for use with 32-bit and 64-bit pointer sizes, respectively.

### Description

The `mempcpy` function, similar to the `memcpy` function, copies *size* bytes from the object pointed to by *source* to the object pointed to by *dest*; it does not check for the overflow of the receiving memory area (*dest*). Instead of returning the value of *dest*, `mempcpy` returns a pointer to the byte following the last written byte.

### Returns

The `mempcpy` function returns a pointer to the byte following the last written byte.

## getline, getwline, getdelim, getwdelim

### Format

```
#include <stdio.h>
ssize_t getline (char **lineptr, size_t *n, FILE *stream);
ssize_t getwline (wchar_t **lineptr, size_t *n, FILE *stream);
ssize_t getdelim (char **lineptr, size_t *n, int delimiter, FILE *stream);
ssize_t getwdelim (wchar_t **lineptr, size_t *n, wint_t delimiter,
FILE *stream);
```

### Function Variants

The `getline` function has variants named `_getline32` and `_getline64` for use with 32-bit and 64-bit pointer sizes, respectively.

The `getwline` function has variants named `_getwline32` and `_getwline64` for use with 32-bit and 64-bit pointer sizes, respectively.

The `getdelim` function has variants named `_getdelim32` and `_getdelim64` for use with 32-bit and 64-bit pointer sizes, respectively.

The `getwdelim` function has variants named `_getwdelim32` and `_getwdelim64` for use with 32-bit and 64-bit pointer sizes, respectively.

### Description

The `getline` and `getwline` functions read an entire line from *stream*, storing the address of the buffer, which contains the text into *\*lineptr*. The buffer is null-terminated and includes the newline character if one was found.

If *\*lineptr* is NULL, then `getline` will allocate a buffer for storing the line, which should be freed by the user program. (In this case, the value in *\*n* is ignored.)

Alternatively, before calling `getline`, *\*lineptr* can contain a pointer to a `malloc` allocated buffer *\*n* bytes in size. If the buffer is not large enough to hold the line, `getline` resizes it with `realloc`, updating *\*lineptr* and *\*n* as necessary.

The `getdelim` and `getwdelim` functions work like `getline` and `getwline`, respectively, except that a line delimiter other than newline can be specified as the delimiter argument. As with `getline` and `getwline` a delimiter character is not added if one was not present in the input before end of file was reached.

### Returns

On success, all functions return the number of characters read, including the delimiter character, but not including the terminating null byte.

## qsort\_r

### Format

```
#include <stdlib.h>
void qsort_r (void *base, size_t nmemb, size_t size,
int (*compar)(const void *, const void *, void *), void *arg)
```

### Function Variants

The `qsort_r` function has variants named `_qsort_r32` and `_qsort_r64` for use with 32-bit and 64-bit pointer sizes, respectively.

### Description

The `qsort_r` function is the reentrant version of `qsort`. See the `qsort` description in the *VSI C User's Guide for OpenVMS Systems*. `qsort_r` is identical to `qsort` except that the comparison function `compar` takes a third argument. A pointer is passed to the comparison function via `arg`.

### Returns

The `qsort_r` function returns no value.

## mkostemp

### Format

```
#include <stdlib.h>
int mkostemp (char *template, int flags)
```

### Description

The `mkostemp` function is equivalent to `mkstemp`, with the difference that flags as for open may be specified in `flags`.

The `mkostemp` function replaces the six trailing Xs of the string pointed to by `template` with a unique set of characters, and returns a file descriptor for the file opened using the flags specified in `flags`.

The string pointed to by `template` should look like a filename with six trailing X's. The `mkostemp` function replaces each X with a character from the portable filename character set, making sure not to duplicate an existing filename.

If the string pointed to by `template` does not contain six trailing Xs, -1 is returned.

### Returns

On success, the `mkostemp` function returns a file descriptor for the open file.

-1 indicates an error. The string pointed to by `template` does not contain six trailing Xs.

## posix\_memalign

### Format

```
#include <stdlib.h>
int posix_memalign (void ** memptr, size_t alignment, size_t size)
```

## Function Variants

The `posix_memalign` function has variants named `_posix_memalign32` and `_posix_memalign64` for use with 32-bit and 64-bit pointer sizes, respectively.

## Description

The `posix_memalign()` function allocates `size` bytes of memory such that the allocation's base address is an exact multiple of `alignment`, and returns the allocation in the value pointed to by `memptr`.

The requested alignment must be a power of 2 at least as large as `sizeof( void * )`. Memory that is allocated via `posix_memalign()` can be used as an argument in subsequent calls to `realloc()` and `free()`.

---

## Note

The allocation returned by `realloc()` is not guaranteed to preserve the original alignment

---

## Returns

The `posix_memalign` function returns 0 if successful, and an error value otherwise.

# aligned\_alloc

## Format

```
#include<stdlib.h>
void * aligned_alloc (size_t alignment, size_t size)
```

## Function Variants

The `aligned_alloc` function has variants named `_aligned_alloc32` and `_aligned_alloc64` for use with 32-bit and 64-bit pointer sizes, respectively.

## Description

The `aligned_alloc` function allocates space for an object whose alignment is specified by `alignment`, whose size is specified by `size`, and whose value is indeterminate. Memory that is allocated via `aligned_alloc()` can be used as an argument in subsequent calls to `realloc()` and `free()`.

---

## Note

The allocation returned by `realloc()` is not guaranteed to preserve the original alignment.

---

## Returns

The `aligned_alloc` function returns a pointer to the allocated memory or NULL if the memory can't be allocated.

# asprintf, vasprintf

## Format

---

```
int asprintf (char **__sp, const char * __format, ...);
int vasprintf (char ** __sp, const char * __format, __va_list__ __arg);
```

## Description

The functions `asprintf()` and `vasprintf()` are mostly similar to `sprintf` and `vsprintf`, except that they allocate a string large enough to hold the output, including the terminating null byte ('\0'), and return a pointer to it via the first argument. This pointer should be passed to `free` to release the allocated storage when it is no longer needed.

## 2. Updates to Functions

- The `open`, `fopen`, and `popen` functions have been updated to support close on exec. The `open` function now supports the `O_CLOEXEC` flag. The `fopen` and `popen` functions now support “e” in the access mode.
- The `fcntl` function has been updated to support the `O_NONBLOCK` flag in the `F_SETFL` and `F_GETFL` modes.
- The `setbuf` and `setvbuf` functions have been updated to take 64-bit arguments.

However, the *buffer* parameter must contain a 32-bit memory buffer, therefore when compiling the application in 64-bit mode with `/POINTER=64` or `/POINTER=LONG`, `_malloc32` must be used to allocate the buffer.

- For `getopt` and `localeconv`, 64-bit function variants (`_getopt64` and `_localeconv64`) have been added.
- The *addrinfo* and *passwd* structures have been updated to work better in 64-bit mode with the `getaddrinfo`, `freeaddrinfo`, `getpwnam`, `getpwuid`, and `getpwent` functions.

Previously, to use the 64-bit versions of *addrinfo* and *passwd*, it was necessary to use `__addrinfo64` and `__passwd64` structures because *addrinfo* and *passwd* were always 32-bit.

Now, when compiling in 64-bit mode with `/POINTER=64` or `/POINTER=LONG`, *addrinfo* and *passwd* structures are correctly compiled as the 64-bit versions, `__addrinfo64` and `__passwd64`. This behavior is similar to other 64-bit structures.

To retain the previous 32-bit behavior of *addrinfo* and *passwd* when compiling in 64-bit mode, you can either replace the *addrinfo* and *passwd* structures with their 32-bit versions, `__addrinfo32` and `__passwd32`, or revert to the previous definitions of these structures by compiling your application with the `/DEFINE=(__CRTL_VER_OVERRIDE = 80400000)` switch.

- The `poll` function has been updated to support pipes, mailboxes, TTYs, and files.
- The arguments to `fwrite()` are now checked to conform to the POSIX standard
- The arguments to the `exec*()` functions are checked to avoid access violation errors when the `argv` parameter is `NULL`
- The `execv`, `execve`, and `execvp` functions have been enhanced to support 64-bit pointers for the `argv` argument
- `O_NONBLOCK` mode can be enabled or disabled for mailboxes and channels

- The `gettim()` function now supports `CLOCK_MONOTONIC`, `CLOCK_MONOTONIC_COARSE`, and `CLOCK_MONOTONIC_RAW`
- Calling the `inet_anon()` function with 64-bit arguments no longer result in an ACCVIO error
- Performance of the `setlocale()` function has been improved
- The functions `writenv()`, `pwrite()`, `write()`, and `fwrite()` are now atomic
- A 64-bit version of `execle()` has been added
- The `iconv()` function now accepts 64-bit pointers
- If the `realpath()` function is called with the `resolved_name` parameter equal to null, the CRTL will allocate a buffer to hold the generated pathname. The user will be responsible for freeing the buffer by calling the `free()` function
- If the `getcwd()` function is called with the `buf` parameter equal to null and the `size` parameter equal to 0 (zero), the CRTL will allocate a buffer to hold the output string. The user will be responsible for freeing the buffer by calling the `free()` function

### 3. Bug Fixes

- The `open` function now works properly when opening `/dev/null` and `/dev/tty` when `DECC$POSIX_COMPLIANT_PATHNAMES` is defined as 1, 2, or 3.
- Multiple processes or multiple threads attempting to open a file for append at the same time now correctly open the same file.
- If the `fopen` function is called with the `O_TRUNC` flag and the file specification includes a file version number, the function truncates the file when open rather than returns an error.
- The `shmget` function can be called a second time with the same key value and a size of 0.
- The `stat` function now returns the correct value for `st_blocks` when the file allocation value is greater than 65536 blocks.
- The `fpclassify` syntax has been fixed in `MATH.H` to compile classification macros correctly.
- The `strptime` function now works properly with the `%Ow` conversion specifier.
- The `unlink` function now works properly when called with a POSIX path but without defining the required DECC\$ feature logical or without specifying the `K_UNIX` argument.
- The `nanosleep` function is now reentrant.
- `MATH$FP_CLASS_<n>X` functions, added as part of the C99 work, have been added to `STARLET.OLB`
- `fopen()` and `open()` correctly create a new version of a file, rather than overwriting the existing one, if the file is opened for trunc (`O_TRUNC`) and the file specification contains a semicolon but no version number
- Writing 0 bytes to a mailbox device now sends an EOF to the mailbox rather than returning an error
- Idle Samba processes no longer execute excessive buffered I/Os per second

- Various processes, including NTP, no longer go into a compute intensive state
- Specifying non-blocking I/O on sockets no longer results in an I/O error when transferring buffers larger than 62696 bytes
- The function `execlle()` no longer causes an ACCVIO when called incorrectly
- Buffer overflows have been fixed in `execl()`, `execle()`, and `execlp()`
- The `realpath()` function no longer returns an error for non-privileged processes that do not have read access to [000000] when `DECC$POSIX_COMPLIANT_PATHNAMES` is defined to 1.
- The `access()` function no longer returns an error when the `file_spec` parameter is set to either `/dev/null` or `/dev/tty`
- The `exec*()` functions no longer leak resources if the call results in an `SS$_EXQUOTA` error
- The `write()` and `pwrite()` functions now return a zero if the length parameter is set to zero. This fixes a problem that was introduced in C RTL ECO V6, where setting the length parameter to zero would result in an error.
- The `getname()` function no longer returns an invalid result in a child process that was created by a parent process using the `exec*` functions
- A buffer overflow has been fixed in `catopen()`
- C RTL ECO V3 introduced a problem in the `wait3()` and `wait4()` functions that could potentially corrupt memory beyond the `rusage` structure of an application. This problem has been fixed in ECO V8.
- The LIBRTL function `LIB$CVT_DX_DX()` no longer returns an incorrect value after receiving the literal 0 as an input

## 4. New Header Files

ALLOCA.H.

PARAMS.H

TERMIOS.H

The macro `va_copy` has been added to `STDARG.H` for Alpha and IA64.

```
#define va_copy(cp, ap) ((cp) = (ap))
```

## 5. Known Limitation

On Integrity, math routines that perform comparisons, with one or both of the parameters being a long double NaN, do not compare correctly.

## 6. Documentation Update

The `sem_open()` function returns a 64-bit pointer to a semaphore, so you must allocate a 64-bit pointer to receive the returned semaphore pointer. One way to do this is as follows:

```
#pragma __required_pointer_size __save
#pragma __required_pointer_size 64
sem_t *mysemp = NULL;
#pragma __required_pointer_size __restore
mysemp = sem_open (...);
```

The action routine called by the `decc$to_vms` function takes an optional third parameter which is a void pointer to an argument that is passed to the action routine. This optional parameter to the action routine is passed as an optional, final argument to `decc$to_vms`. The format for `decc$to_vms` is:

```
#include <unixlib.h>
int decc$to_vms (const char *unix_style_filespec,
int (*action_routine) (char *OpenVMS_style_filespec,
int type_of_file, ...), int allow_wild, int no_directory, ...);
```

The action routine called by the `decc$from_vms` function takes an optional second parameter which is a void pointer to an argument that is passed to the action routine. This optional parameter to the action routine is passed as an optional, final argument to `decc$from_vms`. The format for `decc$from_vms` is:

```
#include <unixlib.h>
int decc$from_vms (const char *vms_filespec,
int (*action_routine) (char *UNIX_style_filespec, ...),
int wild_flag, ...);
```

