VMS Software

VSI OpenVMS

C Run-Time Library (C RTL) Release Notes

Publication Date: July 2022

Operating System and Version: ECO patch kit RTL V6.0

VMS Software, Inc. (VSI)
Burlington, Massachusetts, USA

C Run-Time Library (C RTL) Release Notes

Legal Notice

C RTL Changes in ECO Patch Kit RTL V6.0

The ECO patch kit RTL V6.0 provides additional C RTL functions, updates to some functions, bug fixes, new header files, and identifies a known limitation.

The ECO patch kit RTL V6.0 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

VSI OpenVMS x86-64 Version 9.1-A Field Test and future versions of VSI OpenVMS for x86-64 will contain the C RTL changes implemented in the ECO patch kit RTL V6.0.

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
  #undefine 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 V6.0.

1. New Functions

This section describes the new C RTL functions introduced in the ECO patch kit RTL V6.0.

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* [https://docs.vmssoftware.com/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* [https://vmssoftware.com/docs/VSI_C_USER.pdf]. `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.

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 *__addreinfo64* 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, *__addreinfo64* 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, *__addreinfo32* 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 CLOCL_MONOTONIC_RAW
- Calling `inet_anon()` with 64-bit arguments no longer result in an ACCVIO error

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

4. New Header Files

ALLOCA.H.

PARAMS.H

TERMIOS.H

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.